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

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Mathematical Sciences

ISBN 978-82-471-1713-2 (printed ver.)
ISBN 978-82-471-1714-9 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2009:160

Printed by NTNU-trykk

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

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: π represents the protocol to be executed, $\mathcal{A}$ is the *adversary*, and $\mathcal{Z}$ is the *environment*. While $\mathcal{A}$ represents attacks directly aimed at the instance of π in question, $\mathcal{Z}$ represents the other protocol instances running in the system, including those using the instance in question as a subroutine. Given π, $\mathcal{A}$ and $\mathcal{Z}$, the model is defined as a system of ITMs, where $\mathcal{Z}$ is the initial ITM, and the control function is defined as follows:

- $\mathcal{Z}$ first invokes the adversary $\mathcal{A}$. In following activations, $\mathcal{Z}$ provides inputs either to $\mathcal{A}$ or some instance of π. 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 $\mathcal{Z}$'s subroutine output tape. We note that no restrictions are made on the written messages. $\mathcal{A}$ corrupts a party if and only if instructed to do so by $\mathcal{Z}$, 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 π may write a message to $\mathcal{A}$'s incoming communication tape or $\mathcal{Z}$'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 $\mathcal{Z}$ halts. Without loss of generality, we assume that $\mathcal{Z}$'s output consists of a single bit. The model of protocol execution is illustrated in Figure 1.

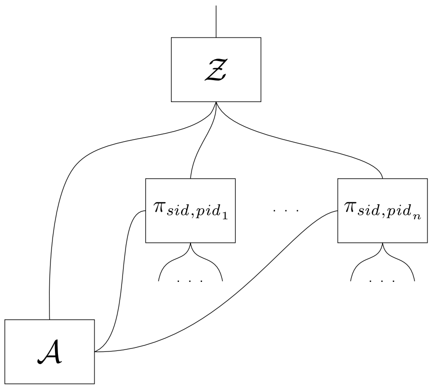

FIGURE 1. The model of protocol execution. For each ITI, a line coming from above represents the input tape, a line coming from below represents the subroutine output tape, and a line coming from either side represents the incoming communication tape.

Regarding the model of protocol execution, note that $\mathcal{Z}$ and $\mathcal{A}$ 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 $\mathcal{A}$, and not directly to each other. This means that $\mathcal{A}$ is in total control of the network. In particular, the messages delivered by $\mathcal{A}$ 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 $\mathcal{A}$, and follows $\mathcal{A}$'s instructions in all remaining activations. Note that, while in general, $\mathcal{Z}$ may instruct $\mathcal{A}$ 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 $\mathcal{Z}$ 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 $\mathrm{IDEAL}_{\mathcal{F}}$, 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 $\mathcal{S}$. The communication between $\mathcal{F}$ and $\mathcal{S}$ represents the influence/information that an adversary is allowed to obtain. For instance, $\mathcal{S}$ 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 $\mathcal{Z}$ when interacting with a protocol π and an adversary $\mathcal{A}$. We denote this ensemble by

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

where n is the *security parameter* and z is the external input of $\mathcal{Z}$.

Definition 1. *A protocol π* UC-emulates *a protocol ϕ if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that*

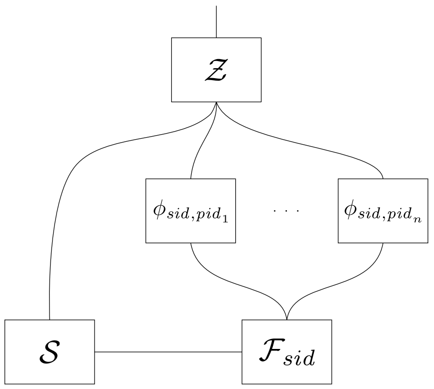

FIGURE 2. The model of execution for an ideal protocol ϕ.

for any PPT environment $\mathcal{Z}$ we have

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

for some negligible function ν.

π UC-realizes *an ideal functionality $\mathcal{F}$ if it UC-emulates the ideal protocol $IDEAL_{\mathcal{F}}$.*

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

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

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

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

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

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

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

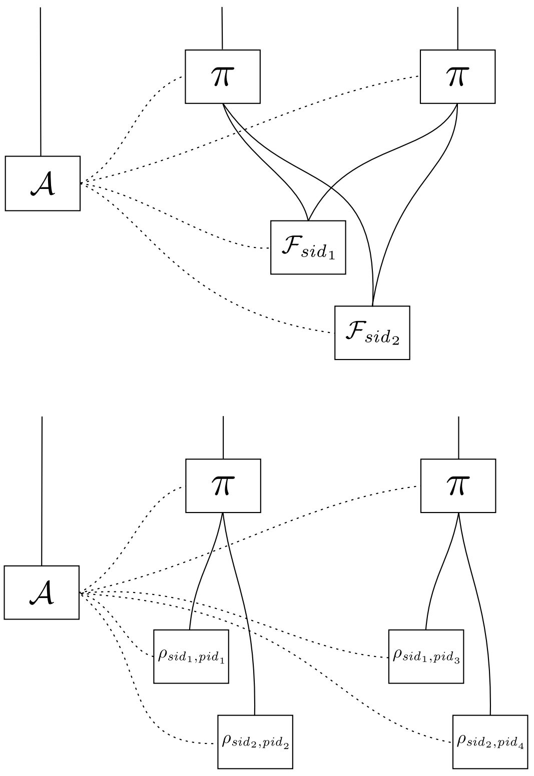

FIGURE 3. The $\mathcal{F}$-hybrid protocol π and the composed protocol $\pi^{\rho/\mathcal{F}}$.

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

[1] Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *CCS '97: Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 36–47, New York, NY, USA, 1997. ACM.

[2] Andrew C. Yao. Protocols for Secure Computation. In *23rd Annual Symp. on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

[3] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A General Composition Theorem for Secure Reactive System. In *Proceedings of 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, February 2004.

[4] Donald Beaver. Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *J. Cryptology*, 4(2):75–122, 1991.

[5] Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In *CRYPTO '93: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, pages 232–249, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[6] Michael Burrows, Martin Abadi, and Roger Needham. A Logic of Authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[7] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Available at `http://eprint.iacr.org/2000/067`.

[8] Ran Canetti. Security and Composition of Cryptographic Protocols: A Tutorial (part i). *SIGACT News*, 37(3):67–92, 2006.

[9] Ran Canetti. Composable Formal Security Analysis: Juggling Soundness, Simplicity and Efficiency. In *ICALP '08: Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, pages 1–13, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] Ran Canetti and Jonathan Herzog. Universally Composable Symbolic Analysis of Cryptographic Protocols (the Case of Encryption-Based Mutual Authentication and Key Exchange). In *2004/334, International Association for Cryptological Research*, 2004.

[11] David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology-Crypto'82*, pages 199–203, 1982.

[12] C. Cremers. *Scyther - Semantics and Verification of Security Protocols.* PhD thesis, Eindhoven University of Technology, 2006.

[13] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[14] Marc Fischlin. Round-Optimal Composable Blind Signatures in the Common Reference String Model. In *Advances in Cryptology-Crypto 2006*. Springer-Verlag, 2006.

[15] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *STOC '87: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[16] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and Systems Sciences*, 28(2):270–299, 1984.

[17] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[18] Shafi Goldwasser and Leonid A. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO*, pages 77–93, 1990.

[19] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of Blind Digital Signatures (Extended Abstract). In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 150–164, London, UK, 1997. Springer-Verlag.

[20] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *TACAs '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147–166, London, UK, 1996. Springer-Verlag.

[21] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996.

[22] Silvio Micali and Phillip Rogaway. Secure Computation (Abstract). In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 392–404, London, UK, 1992. Springer-Verlag.

[23] Waka Nagao, Yoshifumi Manabe, and Tatsuaki Okamoto. A Universally Composable Secure Channel Based on the KEM-DEM Framework. In *TCC*, pages 426–444, 2005.

[24] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, 1978.

[25] Birgit Pfitzmann and Michael Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *CCS '00: Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 245–254, New York, NY, USA, 2000. ACM.

[26] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-Interactive Zero-Knowledge from Homomorphic Encryption. In *TCC*, pages 41–59, 2006.

[27] Brent Waters. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*, pages 114–127, 2005.

[28] Yuliang Zheng. Digital Signcryption or How to Achieve Cost(Signature & Encryption) $\ll$ Cost(Signature) + Cost(Encryption). In *Advances in Cryptology - CRYPTO '97*, pages 165–179, Berlin Heidelberg, 1997. Springer-Verlag.

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 $\mathcal{SC}$ 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 τ (encoded as 1^τ) 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 τ (encoded as 1^τ) 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 $\perp$, 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 $\mathbf{Exp}^{\text{ind-cca2}}_{\mathcal{SC},A}$ 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 $\mathcal{O}_S$, which performs signcryption under the fixed key sk^s and an arbitrary key $pk^{r'}$, and a

```
$\mathbf{Exp}^{\text{ind-cca2}}_{\mathcal{SC},A}(\tau)$
    (1) $(sk^s, pk^s) \leftarrow K_s(\tau)$.
    (2) $(sk^r, pk^r) \leftarrow K_r(\tau)$.
    (3) $(m_0, m_1, state) \leftarrow$
        $A_1^{\mathcal{O}_S,\mathcal{O}_U}(pk^s, pk^r)$.
    (4) $b \leftarrow \{0,1\}$.
    (5) $c \leftarrow S(sk^s, pk^r, m_b)$.
    (6) $b' \leftarrow$
        $A_2^{\mathcal{O}_S,\mathcal{O}_U}(pk^s, pk^r, m_0, m_1, c, state)$.
    (7) If $b' = b$ then return 1, otherwise
        return 0.

$\mathbf{Exp}^{\text{ext-cma}}_{\mathcal{SC},A}(\tau)$
    (1) $(sk^s, pk^s) \leftarrow K_s(\tau)$.
    (2) $(sk^r, pk^r) \leftarrow K_r(\tau)$.
    (3) $c \leftarrow A^{\mathcal{O}_S,\mathcal{O}_U}(pk^s, pk^r)$.
    (4) If $U(pk^s, sk^r, c) \neq \perp$
        then return 1, otherwise
        return 0.

$\mathbf{Exp}^{\text{ror-cca2}}_{\mathcal{SC},A}(\tau)$
    (1) $(sk^s, pk^s) \leftarrow K_s(\tau)$
    (2) $(sk^r, pk^r) \leftarrow K_r(\tau)$
    (3) $b \leftarrow \{0,1\}$
    (4) $b' \leftarrow$
        $A^{\mathcal{O}_S,\mathcal{O}_U,\mathcal{O}^b_{\text{ror}}}(pk^s, pk^r)$
    (5) Return $b'$.
```

FIGURE 1. Experiments for security definitions.

flexible unsigncryption oracle $\mathcal{O}_U$, which performs unsigncryption under an arbitrary key $pk^{s'}$ and the fixed key sk^r. A_2 is not allowed to query $\mathcal{O}_U$ with the ciphertext c and the sender key pk^s.

A is said to win if the experiment returns 1. We define the *advantage* of A in breaking $\mathcal{SC}$ with respect to IND-CCA2 as

$$\mathbf{Adv}^{\text{ind-cca2}}_{\mathcal{SC},A}(\tau) = \left|2 \cdot \Pr\left[\mathbf{Exp}^{\text{ind-cca2}}_{\mathcal{SC},A}(\tau) = 1\right] - 1\right|.$$

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

The second experiment $\mathbf{Exp}^{\text{ext-cma}}_{\mathcal{SC},A}(\tau)$ 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

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

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

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 $\mathbf{Exp}^{\text{ror-cca2}}_{\mathcal{SC},A}(\tau)$, the adversary A has access to the oracle $\mathcal{O}^b_{\text{ror}}$ (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 $\mathcal{O}_S$ and $\mathcal{O}_U$.

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

We define the *advantage* of A in breaking $\mathcal{SC}$ with respect to ROR-CCA2 as

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

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

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

Theorem 2. *A signcryption scheme $\mathcal{SC}$ 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 $\mathcal{F}_{\text{SC}}$ 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 $\mathcal{F}_{\text{SM}}$ 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 π_{SM} that securely realizes $\mathcal{F}_{\text{SM}}$ in the $(\mathcal{F}_{\text{CA}}, \mathcal{F}_{\text{SC}})$-hybrid model, where $\mathcal{F}_{\text{CA}}$ 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 $\mathcal{F}_{\text{CA}}$ is defined in Figure 2, and is similar to the one given by Canetti in [2]. Next, in Figure 3, the ideal signcryption functionality $\mathcal{F}_{\text{SC}}$ is defined, and then, in Figure 4, the ideal secure messaging functionality $\mathcal{F}_{\text{SM}}$ is defined. The protocol π_{SM} is described in Figure 5.

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

Theorem 3. *The protocol π_{SM} securely realizes $\mathcal{F}_{SM}$ in the $(\mathcal{F}_{CA}, \mathcal{F}_{SC})$-hybrid model.*

Functionality $\mathcal{F}_{\text{CA}}$

$\mathcal{F}_{\text{CA}}$ proceeds as follows, with parties $P_1, \dots, P_n$ and an ideal adversary $\mathcal{S}$.

CA.Register

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

CA.Retrieve

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

FIGURE 2. The ideal certification authority functionality $\mathcal{F}_{\text{CA}}$.

4. SECURELY REALIZING $\mathcal{F}_{\text{SC}}$

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

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

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

Finally, we prove the following result:

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

Proof. Assume that $\mathcal{SC}$ 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 $\mathcal{A}$ interacting with parties running $\pi_{\mathcal{SC}}$, there is an ideal adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and $\pi_{\mathcal{SC}}$ or with $\mathcal{S}$ and the ideal protocol for $\mathcal{F}_{\text{SC}}$, $\text{IDEAL}_{\mathcal{F}_{\text{SC}}}$.

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

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

Simulating SC.KeyGen: Upon receiving the first message $(\textbf{SC.KeyGen}, sid)$ from some party P_i, $\mathcal{F}_{\text{SC}}$ sends $(\textbf{SC.KeyGen}, sid, P_i)$ to $\mathcal{S}$. $\mathcal{S}$ 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 $(\textbf{SC.Key}, sid, P_i, (pk_i^s, pk_i^r))$ to $\mathcal{F}_{\text{SC}}$.

Functionality $\mathcal{F}_{\text{SC}}$

$\mathcal{F}_{\text{SC}}$ proceeds as follows, with parties $P_1, \dots, P_n$ and an ideal adversary $\mathcal{S}$.

Upon receiving a message from a corrupt party, $\mathcal{F}_{\text{SC}}$ forwards the message to $\mathcal{S}$, and when $\mathcal{S}$ replies to this message, $\mathcal{F}_{\text{SC}}$ forwards the reply to the corrupt party.

SC.KeyGen

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

SC.Encrypt

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

(1) If $pk^r = pk_j^r$ for some j and P_j is honest, then send ($\mathbf{SC.Encrypt}, sid, pk_i^s, pk^r, |m|$) to $\mathcal{S}$. Otherwise send ($\mathbf{SC.Encrypt}, sid, pk_i^s, pk^r, m$) to $\mathcal{S}$.
(2) Upon receiving ($\mathbf{SC.Ciphertext}, sid, pk_i^s, pk^r, c$) from $\mathcal{S}$ such that there is no recorded entry (pk_i^s, pk^r, m', c) for any m', output ($\mathbf{SC.Ciphertext}, sid, pk^r, m, c$) 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^r, m, c).

SC.Decrypt

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

(1) Send ($\mathbf{SC.Decrypt}, sid, pk^s, pk_j^r, c$) to $\mathcal{S}$. Upon receiving ($\mathbf{SC.Plaintext}, sid, pk^s, pk_j^r, m'/\perp, c$) from $\mathcal{S}$, continue.
(2) If an entry (pk^s, pk_j^r, m, c) is recorded, then output ($\mathbf{SC.Plaintext}, sid, pk^s, m, c$) to P_j.
(3) Otherwise, if $pk^s = pk_i^s$ for some i and P_i is honest, then output ($\mathbf{SC.Plaintext}, sid, pk^s, \perp, c$) to P_j.
(4) Otherwise, output ($\mathbf{SC.Plaintext}, sid, pk^s, m'/\perp, c$) to P_j.

FIGURE 3. The ideal signcryption functionality $\mathcal{F}_{\text{SC}}$.

Simulating SC.Encrypt: Assume that $\mathcal{F}_{\text{SC}}$ receives ($\mathbf{SC.Encrypt}, sid, pk^r, m$) 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) $\mathcal{F}_{\text{SC}}$ sends ($\mathbf{SC.Encrypt}, sid, pk_i^s, pk^r, |m|$) to $\mathcal{S}$. $\mathcal{S}$ 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 ($\mathbf{SC.Ciphertext}, sid, pk_i^s, pk^r, c$) to $\mathcal{F}_{\text{SC}}$, and records (pk_i^s, pk^r, c).
(2) $\mathcal{F}_{\text{SC}}$ sends ($\mathbf{SC.Encrypt}, sid, pk_i^s, pk^r, m$) to $\mathcal{S}$. $\mathcal{S}$ encrypts m using the signcryption function S with sk_i^s and pk^r. He obtains c, and checks

Functionality $\mathcal{F}_{\text{SM}}$

$\mathcal{F}_{\text{SM}}$ proceeds as follows, with parties $P_1, \dots, P_n$ and an ideal adversary $\mathcal{S}$.

Upon receiving a message from a corrupt party, $\mathcal{F}_{\text{SM}}$ forwards the message to $\mathcal{S}$, and when $\mathcal{S}$ replies to this message, $\mathcal{F}_{\text{SM}}$ forwards the reply to the corrupt party.

SM.Register

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

SM.Encrypt

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

(1) If there is a recorded entry P_i, then continue. Otherwise, output $(\textbf{SM.Encrypt.Error}, \mathit{sid}, P_i \textit{ not registered})$ to P_i.
(2) If P_j is honest, then send $(\textbf{SM.Encrypt}, \mathit{sid}, P_i, P_j, |m|)$ to $\mathcal{S}$. Otherwise, send $(\textbf{SM.Encrypt}, \mathit{sid}, P_i, P_j, m)$ to $\mathcal{S}$.
(3) Upon receiving $(\textbf{SM.Ciphertext}, \mathit{sid}, P_i, P_j, c)$ from $\mathcal{S}$ 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 $(\textbf{SM.Encrypt.Error}, \mathit{sid}, P_j \textit{ not registered})$ to P_i.
(4) Output $(\textbf{SM.Ciphertext}, \mathit{sid}, P_j, m, c)$ to P_i. If P_j is honest, then record the entry (P_i, P_j, m, c).

SM.Decrypt

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

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

FIGURE 4. The ideal secure messaging functionality $\mathcal{F}_{\text{SM}}$.

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 $(\textbf{SC.Ciphertext}, \mathit{sid}, pk_i^s, pk^r, c)$ to $\mathcal{F}_{\text{SC}}$.

Protocol π_{SM}

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

SM.Register

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

SM.Encrypt

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

SM.Decrypt

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

FIGURE 5. The secure messaging protocol π_{SM}.

Simulating SC.Decrypt: Upon receiving ($\mathbf{SC.Decrypt}, sid, pk^s, c$) from some party P_j, $\mathcal{F}_{\mathrm{SC}}$ sends ($\mathbf{SC.Decrypt}, sid, pk^s, pk_j^r, c$) 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 ($\mathbf{SC.Plaintext}, sid, pk^s, pk_j^r, m', c$) to $\mathcal{F}_{\mathrm{SC}}$.
(2) $\mathcal{S}$ decrypts c using the unsigncryption function U with pk^s and sk_j^r, obtains $m'/\perp$ and sends ($\mathbf{SC.Plaintext}, sid, pk^s, pk_j^r, m'/\perp, c$) to $\mathcal{F}_{\mathrm{SC}}$.

Protocol $\pi_{\mathcal{SC}}$

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

SC.KeyGen

Upon the first input $(\mathbf{SC.KeyGen}, sid)$, 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 $(\mathbf{SC.Key}, sid, (pk_i^s, pk_i^r))$.

SC.Encrypt

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

SC.Decrypt

Upon input $(\mathbf{SC.Decrypt}, sid, pk^s, c)$, P_i obtains $m/\perp = U(pk^s, sk_i^r, c)$ and outputs $(\mathbf{SC.Plaintext}, sid, pk^s, m/\perp, c)$.

FIGURE 6. The signcryption protocol $\pi_{\mathcal{SC}}$.

It is clear that $\mathcal{S}$ simulates **SC.KeyGen** perfectly.

As for **SC.Encrypt**, if P_i and P_j are honest, and $pk^r = pk_j^r$ for some j, we observe that in $\pi_{\mathcal{SC}}$, the real message m is encrypted, while in IDEAL$_{\mathcal{F}_{\mathrm{SC}}}$, a random message m' is encrypted. Moreover, in IDEAL$_{\mathcal{F}_{\mathrm{SC}}}$, for a fixed key pair (pk^s, pk^r), a message will never be encrypted into the same ciphertext twice. In $\pi_{\mathcal{SC}}$ 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 $pk^s \neq pk_i^s$ for all i. Otherwise, the following separates $\pi_{\mathcal{SC}}$ from IDEAL$_{\mathcal{F}_{\mathrm{SC}}}$: In $\pi_{\mathcal{SC}}$, the decryption of a ciphertext c is determined by the unsigncryption function U, while in IDEAL$_{\mathcal{F}_{\mathrm{SC}}}$, the decryption of c is determined according to whether there is a recorded tuple (pk_i^s, pk_j^r, m, c). If there is, then m is returned, otherwise the symbol $\perp$ 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 IDEAL$_{\mathcal{F}_{\mathrm{SC}}}$ and in $\pi_{\mathcal{SC}}$. Otherwise, there are two scenarios to consider: If c is an invalid encryption according to the unsigncryption function U, then the output will be $\perp$ both in IDEAL$_{\mathcal{F}_{\mathrm{SC}}}$ and $\pi_{\mathcal{SC}}$. However, if c is valid, the decryption of c according to U will be output in $\pi_{\mathcal{SC}}$, while $\perp$ will be output in IDEAL$_{\mathcal{F}_{\mathrm{SC}}}$.

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

We want to show that if there exists an environment $\mathcal{Z}$ able to distinguish IDEAL$_{\mathcal{F}_{\mathrm{SC}}}$ and $\pi_{\mathcal{SC}}$ with non-negligible probability, then we can construct an adversary A^{ror} breaking the ROR-CCA2 security or an adversary A^{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, $\mathcal{Z}$ interacts with the protocol $\pi_{\mathcal{SC}}$ running with parties $P_1, \dots, P_n$.

Game 1: This game is the same as Game 0, with the following modifications: The honest parties $P_1, \dots, P_t$ 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 $\mathcal{Z}$, exactly as P_i would do in the protocol $\pi_{\mathcal{SC}}$. P records $(P_i, sk_i^s, pk_i^s, sk_i^r, pk_i^r)$ 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 $\mathcal{Z}$'s point of view, there is no difference between Game 0 and Game 1. Therefore, if we let G_i denote the output of $\mathcal{Z}$ when taking part in Game i, we have

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

Game 2: This game is the same as Game 1, with the following modification: When receiving messages $(\textbf{SC}.\textbf{Decrypt}, sid, pk_i^s, c)$ 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 $(\textbf{SC}.\textbf{Plaintext}, sid, pk_i^s, m, c)$. Otherwise, P outputs $(\textbf{SC}.\textbf{Plaintext}, sid, pk_i^s, \perp, c)$.

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

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

$$\Big|Pr[G_1 = 1] - Pr[G_2 = 1]\Big| \le t(t-1)\boldsymbol{Succ}^{ext\text{-}cma}_{\mathcal{SC},A^{ext}}(\tau).$$

Game 3: This game is the same as Game 2, with the following modification: When P receives a message $(\textbf{SC}.\textbf{Encrypt}, sid, pk_2^r, m)$ 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 $(\textbf{SC}.\textbf{Ciphertext}, sid, pk_2^r, m, c)$ on the output tape corresponding to P_1, and records (pk_1^s, pk_2^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 $2 \le i \le t(t-1)+1$, there is an adversary A^{ror} such that*

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

Game $t(t-1)+3$: In this game, $\mathcal{Z}$ interacts with the ideal protocol $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{SC}}}$ running with parties $P_1, \dots, P_n$.

From $\mathcal{Z}$'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*

$$Pr[c = c' | c, c' \leftarrow S(sk^s, pk^r, m)] \leq 2\boldsymbol{Adv}^{ror\text{-}cca2}_{\mathcal{SC}, A^{ror}}(\tau).$$

Hence we have

$$\Big|\Pr[G_{t(t-1)+2} = 1] - \Pr[G_{t(t-1)+3} = 1]\Big| = \delta(\tau),$$

for some negligible function $\delta(\tau)$. Finally, the triangle inequality gives

$$\begin{aligned}\Big|\Pr[G_0 = 1] - \Pr[G_{t(t-1)+3} = 1]\Big| \leq\ & \Big|\Pr[G_0 = 1] - \Pr[G_1 = 1]\Big| + \\ & \Big|\Pr[G_1 = 1] - \Pr[G_2 = 1]\Big| + \cdots + \\ & \Big|\Pr[G_{t(t-1)+2} = 1] - \Pr[G_{t(t-1)+3} = 1]\Big| \\ =\ & t(t-1)\mathbf{Succ}^{\text{ext-cma}}_{\mathcal{SC}, A^{\text{ext}}}(\tau) + t(t-1)\mathbf{Adv}^{\text{ror-cca2}}_{\mathcal{SC}, A^{\text{ror}}}(\tau) + \delta(\tau).\end{aligned}$$

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

5. Concluding Remarks

We have proposed ideal functionalities for signcryption and secure messaging, and described a protocol π_{SM} that securely realizes $\mathcal{F}_{\text{SM}}$ in the $(\mathcal{F}_{\text{CA}}, \mathcal{F}_{\text{SC}})$-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 $\mathcal{F}_{\text{SC}}$. 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 $\mathcal{F}_{\text{SC}}$ 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 $\mathcal{F}_{\text{SC}}$ 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 $\mathcal{F}'_{\text{SC}}$ with adaptive corruption.

References

[1] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 394–403, Washington, DC, USA, 1997. IEEE Computer Society.

[2] Ran Canetti. Universally Composable Signature, Certification, and Authentication. Cryptology ePrint Archive, Report 2003/239, 2004. Available at `http://eprint.iacr.org/2003/239`.

[3] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Available at `http://eprint.iacr.org/2000/067`.

[4] Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-Secure, Non-Interactive Public-Key Encryption. Cryptology ePrint Archive, Report 2004/317, 2004. `http://eprint.iacr.org/`.

[5] John Charles Malone-Lee. *On the Security of Signature Schemes and Signcryption Schemes*. PhD thesis, University of Bristol, 2004.

[6] Philip Rogaway. Symmetric Encryption. ECS 227 - Modern Cryptography - Winter 99, 1999. Available at `http://www.cs.ucdavis.edu/~rogaway/classes/227/winter99/`.

[7] Yuliang Zheng. Digital Signcryption or How to Achieve Cost(Signature & Encryption) $\ll$ Cost(Signature) + Cost(Encryption). In *Advances in Cryptology - CRYPTO '97*, pages 165–179, Berlin Heidelberg, 1997. Springer-Verlag.

Appendix A. Proof of Theorem 3

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

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

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

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

$\mathcal{F}_{\text{SM}}$ sends (**SM.Encrypt**, $sid, P_i, P_j, |m|$) to $\mathcal{S}$. $\mathcal{S}$ sends (**CA.Retrieve**, sid, P_j, P_i) to $\mathcal{A}$ in the name of $\mathcal{F}_{\text{CA}}$, and waits for an (**Ok**, sid, P_j, P_i) from $\mathcal{A}$. $\mathcal{S}$ 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 (**SM.Ciphertext**, sid, P_i, P_j, c) to $\mathcal{F}_{\text{SM}}$. Otherwise, the following procedure takes place:

$\mathcal{S}$ sends (**SC.Encrypt**, $sid, pk_i^s, pk_j^r, |m|$) to $\mathcal{A}$ in the name of $\mathcal{F}_{\text{SC}}$. Upon receiving (**SC.Ciphertext**, sid, pk_i^s, pk_j^r, c) from $\mathcal{A}$, $\mathcal{S}$ sends (**SM.Ciphertext**, sid, P_i, P_j, c) to $\mathcal{F}_{\text{SM}}$.

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

$\mathcal{F}_{\text{SM}}$ sends (**SM.Decrypt**, sid, P_i, P_j, c) to $\mathcal{S}$. $\mathcal{S}$ sends (**CA.Retrieve**, sid, P_i, P_j) to $\mathcal{A}$ in the name of $\mathcal{F}_{\text{CA}}$, and waits for an (**Ok**, sid, P_i, P_j) from $\mathcal{A}$. $\mathcal{S}$ checks if there is a recorded entry $(P_i, (pk_i^s, pk_i^r))$. If there is no such entry, he sends $(\mathbf{SM.Plaintext}, sid, P_i, P_j, \perp)$ to $\mathcal{F}_{\mathrm{SM}}$. Otherwise, the following procedure takes place:

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

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

Appendix B. Proof of Theorem 4

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

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

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

No parties are corrupted, and $\mathcal{A}$ is never activated by $\mathcal{Z}$. Since there is no communication between parties, $\mathcal{A}$ never gets any input.

$\mathcal{Z}$ proceeds as follows:

(1) $\mathcal{Z}$ chooses two parties P_i and P_j, sends the message $(\mathbf{SC.KeyGen}, sid)$ to each of them, and obtains key pairs (pk_i^s, pk_i^r) and (pk_j^s, pk_j^r).
(2) $\mathcal{Z}$ chooses $b \in \{0,1\}$ at random and runs A^{ror} with input (pk_i^s, pk_j^r). He answers A^{ror}'s queries to $\mathcal{O}_S/\mathcal{O}_U/\mathcal{O}_b$ by sending appropriate messages to $P_i/P_j/P_i$. In particular, if $b=0$, $\mathcal{Z}$ chooses a random message of appropriate length and asks P_i to encrypt this message. If $b=1$, $\mathcal{Z}$ asks P_i to encrypt the message in A^{ror}'s query.
(3) Finally, when A^{ror} outputs b', $\mathcal{Z}$ outputs 1 if $b'=b$ and 0 otherwise.

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

When $\mathcal{Z}$ interacts with $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{SC}}}$, A^{ror} does not run in its expected environment, but whatever happens, A^{ror} gets no information about b, and $\mathcal{Z}$ outputs 1 with probability $\frac{1}{2}$.

Let $\mathcal{Z}_{\pi_{\mathcal{SC}}}$ and $\mathcal{Z}_{\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{SC}}}}$ denote $\mathcal{Z}$'s output when interacting with $\pi_{\mathcal{SC}}$ and $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{SC}}}$, respectively. The above argument gives

$$\begin{aligned}
& \Big|\Pr[\mathcal{Z}_{\pi_{\mathcal{SC}}} = 1] - \Pr[\mathcal{Z}_{\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{SC}}}} = 1]\Big| \\
= & \Big|\Pr[\mathcal{Z}_{\pi_{\mathcal{SC}}} = 1|b = 1] \cdot \Pr[b = 1] + \Pr[Z_{\pi_{\mathcal{SC}}} = 1|b = 0] \cdot \Pr[b = 0] - \frac{1}{2}\Big| \\
= & \Big|\frac{1}{2}\left(\Pr[\mathcal{Z}_{\pi_{\mathcal{SC}}} = 1|b = 1] + (1 - \Pr[Z_{\pi_{\mathcal{SC}}} = 0|b = 0]) - 1\right)\Big| \\
= & \frac{1}{2}\Big|(\Pr[\mathcal{Z}_{\pi_{\mathcal{SC}}} = 1|b = 1] - \Pr[Z_{\pi_{\mathcal{SC}}} = 0|b = 0])\Big| \\
= & \frac{1}{2}\delta(\tau).
\end{aligned}$$

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

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

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

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

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

$\mathcal{Z}$ proceeds as follows:

(1) $\mathcal{Z}$ chooses two parties P_i and P_j, sends the message $(\mathbf{SC.KeyGen}, sid)$ to each of them, and obtains key pairs (pk_i^s, pk_i^r) and (pk_j^s, pk_j^r).
(2) $\mathcal{Z}$ runs A^{ext} with input (pk_i^s, pk_j^r). He answers A^{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^{ext} outputs c, $\mathcal{Z}$ sends $(\mathbf{SC.Decrypt}, sid, pk_i^s, c)$ to P_j. If P_j outputs a message m, $\mathcal{Z}$ outputs 1. Otherwise, if P_j outputs the symbol $\perp$, $\mathcal{Z}$ outputs 0.

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

$$\begin{aligned}
\Big|\Pr[\mathcal{Z}_{\pi_{\mathcal{SC}}} = 1] - \Pr[\mathcal{Z}_{\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{SC}}}} = 1]\Big| &= \Big|\Pr\left[\mathbf{Exp}^{\text{ext-cma}}_{\mathcal{SC},A}(\tau) = 1\right] - 0\Big| \\
&= \delta(\tau).
\end{aligned}$$

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 $(\mathbf{SC}.\mathbf{Encrypt}, sid, pk^r_j, m)$ sent to P_i.

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

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

We now construct an adversary A^{ext} trying to break the EXT-CMA security of our signcryption scheme. A^{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^{ext} runs simulated copies of $\mathcal{Z}$, $\mathcal{A}$ and the corrupt parties. A^{ext} also simulates P, and behaves exactly as P, with the following exceptions: A^{ext} chooses random numbers $k, l \in \{1, \dots, t\}$ 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^{ext} lets $pk^s_k = pk^s$, runs K_r to obtain (sk^r_k, pk^r_k), outputs (pk^s_k, pk^r_k) and records $(P_k, \cdot, pk^s_k, sk^r_k, pk^r_k)$. When P_l is asked to generate keys, a corresponding procedure takes place, and A^{ext} records $(P_l, sk^s_l, pk^s_l, \cdot, pk^r_l)$. When receiving a message $(\mathbf{SC}.\mathbf{Encrypt}, sid, pk^r_j, m)$ on the input tape corresponding to P_k, A^{ext} answers using its flexible signcryption oracle. Similarly, when receiving a message $(\mathbf{SC}.\mathbf{Decrypt}, sid, pk^r_i, c)$ on the input tape corresponding to P_l, A^{ext} answers using its flexible unsigncryption oracle.

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

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

Hence we have

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

and the proof is complete.

Appendix D. Proof of Lemma 7

Using Game 2 and Game 3, we will construct an adversary A^{ror} trying to break the ROR-CCA2 security of our signcryption scheme. A^{ror} has been given a target sender's public key pk^s and a target receiver's public key pk^r. A^{ror} runs simulated copies of $\mathcal{Z}$, $\mathcal{A}$ and the corrupt parties. A^{ror} also simulates P, and behaves exactly as P, with the following exceptions: A^{ror} lets P_1 be the target sender and P_2 the target receiver. Thus, when P_1 is asked to generate keys, A^{ror} lets $pk^s_1 = pk^s$, runs K_r to obtain (sk_1^r, pk_1^r), outputs (pk_1^s, pk_1^r) and records $(P_1, \cdot, pk_1^s, sk_1^r, pk_1^r)$. Correspondingly, when P_2 is asked to generate keys, A^{ror} lets $pk_2^r = pk^r$, runs K_s to obtain (sk_2^s, pk_2^s), outputs (pk_2^s, pk_2^r) and records $(P_2, sk_2^s, pk_2^s, \cdot, pk_2^r)$. When receiving a message $(\textbf{SC}.\textbf{Encrypt}, sid, pk_2^r, m)$ on the input tape corresponding to P_1, A^{ror} answers using its real-or-random oracle. Moreover, when receiving a message $(\textbf{SC}.\textbf{Encrypt}, sid, pk_j^r, m)$ on the same input tape, and $j \neq 2$, A^{ror} answers using its flexible signcryption oracle. In addition, when receiving a message $(\textbf{SC}.\textbf{Decrypt}, sid, pk_i^s, c)$ on the input tape corresponding to P_2, and P_i is a corrupt party, A^{ror} answers using its flexible unsigncryption oracle. Finally, A^{ror} copies $\mathcal{Z}$'s output to its own output tape.

We observe that when the real-or-random oracle encrypts the real message, A^{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^{ror} as follows:

$$\begin{aligned}\textbf{Adv}^{\text{ror-cca2}}_{\mathcal{SC},A^{\text{ror}}}(\tau) &= \Big|\Pr\big[\textbf{Exp}^{\text{ror-cca2}}_{\mathcal{SC},A^{\text{ror}}}(\tau)=1|b=1\big] - \Pr\big[\textbf{Exp}^{\text{ror-cca2}}_{\mathcal{SC},A^{\text{ror}}}(\tau)=1|b=0\big]\Big| \\ &= \Big|\Pr[G_2=1]-\Pr[G_3=1]\Big|.\end{aligned}$$

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

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

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^{ror} trying to break the ROR-CCA2 security of our signcryption scheme. Upon input (pk^s, pk^r), A^{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^{ror} outputs 1, otherwise A^{ror} outputs 0. Assume that m has length l. The advantage of A^{ror} is then given by

$$\begin{aligned}\textbf{Adv}^{\text{ror-cca2}}_{\mathcal{SC},A^{\text{ror}}}(\tau) &= \Big|\Pr\big[\textbf{Exp}^{\text{ror-cca2}}_{\mathcal{SC},A^{\text{ror}}}(\tau)=1|b=1\big] - \Pr\big[\textbf{Exp}^{\text{ror-cca2}}_{\mathcal{SC},A^{\text{ror}}}(\tau)=1|b=0\big]\Big| \\ &= \Big|\delta(\tau) - \frac{1}{2^l}\delta(\tau)\Big| \\ &= \frac{2^l-1}{2^l}\delta(\tau) \\ &\ge \frac{1}{2}\delta(\tau).\end{aligned}$$

So we have

$$\Pr[c = c' | c, c' \leftarrow S(sk^s, pk^r, m)] \leq 2\mathbf{Adv}^{\text{ror-cca2}}_{\mathcal{SC}, A^{\text{ror}}}(\tau),$$

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 GJØSTEEN, 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* *theorem* then guarantees security. We refer to [2] 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 [5] addresses this question in the context of blind signatures. The author defines an ideal functionality for blind signatures, $\mathcal{F}_{\mathrm{BlSig}}$, and shows that blind signature schemes realizing $\mathcal{F}_{\mathrm{BlSig}}$ in the plain model do not exist. He does this by showing that $\mathcal{F}_{\mathrm{BlSig}}$ can be used to realize the functionality $\mathcal{F}_{\mathrm{com}}$ for commitment schemes, and then applying a well-known impossibility result [3]. To realize $\mathcal{F}_{\mathrm{BlSig}}$, 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 $\mathcal{F}_{\mathrm{com}}$, 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 $\mathcal{BS}$ is a four-tuple (Signer, User, Gen, Verify) with the following properties:*

- *Gen is a probabilistic polynomial time algorithm, which takes as input a security parameter τ (encoded as 1^τ) and outputs a pair (sk, pk) of secret and public keys.*
- *Signer and User are a pair of probabilistic polynomial time interactive Turing machines, given as common input a public key pk. In addition, Signer is given a corresponding secret key sk, and User is given a message m. The length of all inputs must be polynomial in the security parameter τ. Signer and User interact according to the protocol. At the end of the interaction, Signer outputs either completed or not completed and User outputs either fail or $\sigma(m)$.*
- *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 Gen, if both Signer and User follow the protocol, then Signer(sk, pk) outputs completed, User(pk, m) outputs $\sigma(m)$, and 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 $\mathbf{Exp}^{wb}_{\mathcal{BS},A}(\tau)$ (steps 1, 2, ..., 6) in Figure 1, where A is an algorithm which controls Signer but not User. We define the advantage of A in breaking $\mathcal{BS}$ with respect to weak blindness as*

$$\boldsymbol{Adv}^{wb}_{\mathcal{BS},A}(\tau) = \Big| Pr[b' = 1 | b = 1] - Pr[b' = 1 | b = 0] \Big| .$$

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

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

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

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

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 $\mathbf{Exp}^{nf}_{\mathcal{BS},A}(\tau)$ in Figure 1, where A is an algorithm which controls User but not Signer. We define the success rate of A in breaking $\mathcal{BS}$ with respect to non-forgeability as*

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

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

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.)

$\mathbf{Exp}^{\text{wb/b}}_{\mathcal{BS},A}(\tau)$:

1. $(sk, pk) \leftarrow Gen(1^\tau)$. Run A on input $(1^\tau, sk, pk)$.

1'. Run A on input 1^τ. $(sk, pk) \leftarrow A$.

2. $(m_0, m_1) \leftarrow A$.
3. $b \leftarrow \{0, 1\}$.
4. Let A engage in two parallel interactive protocols, the first with $User(pk, m_b)$ and the second with $User(pk, m_{1-b})$.
5. 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.
6. A outputs a bit b'.

$\mathbf{Exp}^{\text{wror/ror}}_{\mathcal{BS},A}(\tau)$:

1. $(sk, pk) \leftarrow Gen(1^\tau)$. Run A on input $(1^\tau, sk, pk)$.

1' Run A on input 1^τ. $(sk, pk) \leftarrow A$.

2. $b \leftarrow \{0, 1\}$.
3. A polynomial (in τ) number of times, A is allowed to output a message m_1:

   If $b = 0$, choose a random message m_0 and let A engage in a protocol with $User(pk, m_0)$. Run a protocol between $Signer(sk, pk)$ and $User(pk, m_1)$ to get $\sigma(m_1)$. If $User(pk, m_0)$ outputs $\sigma(m_0)$, give $\sigma(m_1)$ to A as additional input.

   If $b = 1$, let A engage in a protocol with $User(pk, m_1)$. If $User(pk, m_1)$ outputs $\sigma(m_1)$, give $\sigma(m_1)$ to A as additional input.
4. A outputs a bit b'.

$\mathbf{Exp}^{\text{nf}}_{\mathcal{BS},A}(\tau)$:

1. $(sk, pk) \leftarrow Gen(1^\tau)$.
2. Let $A(1^\tau, pk)$ engage in polynomially many (in τ) parallel interactive protocols, with polynomially many (in τ) copies of $Signer(sk, pk)$, where A decides in an adaptive manner when to stop. Let l be the number of executions, where the *Signer* outputs *completed*.
3. A outputs a collection $\{(m_1, \sigma(m_1)), \ldots, (m_k, \sigma(m_k))\}$, subject to the constraint that $(m_i, \sigma(m_i)) \neq (m_j, \sigma(m_j))$ for $1 \leq i < j \leq k$, and $Verify(pk, m_i, \sigma(m_i))$ outputs *accept* for $1 \leq i \leq k$.

FIGURE 1. Experiments for defining blindness and non-forgeability.

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

$$\boldsymbol{Adv}^{wror/ror}_{\mathcal{BS},A}(\tau) = \Big| Pr[b' = 1 | b = 1] - Pr[b' = 1 | b = 0] \Big|.$$

The scheme $\mathcal{BS}$ is said to be secure with respect to real-or-random blindness if, for all probabilistic polynomial time A, $\boldsymbol{Adv}^{wror/ror}_{\mathcal{BS},A}(\tau)$ is negligible in τ.

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 $\mathcal{BS}$ 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, $\mathcal{F}_{\text{BS}}$, is defined in Figure 2. The protocol $\pi_{\mathcal{BS}}$ 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 $\mathcal{S}$ 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 σ on m. $\pi(m, \sigma)$ outputs 1 if σ 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 $\mathcal{S}$ delay messages between parties. $\mathcal{S}$ also decides when to inform the respective parties about the outcome of a signature request. This allows $\mathcal{S}$ 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 $\mathcal{F}_{\text{BlSig}}$, and obtains a valid pair (m', σ). Upon verification, (m', σ) will be rejected by $\mathcal{F}_{\text{BlSig}}$, 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, σ) may be accepted upon verification, even if $\mathcal{F}_{\text{BS}}$ has never seen m before. We argue that $\mathcal{F}_{\text{BS}}$ 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.

$\mathcal{F}_{\mathrm{BS}}$ proceeds as follows, with signer $\tilde{Q}$, users $\tilde{P}_1, \dots, \tilde{P}_n$ and an ideal adversary $\mathcal{S}$.

On message $(\textbf{KeyGen})$ from the signer $\tilde{Q}$:

1. Send $(\textbf{KeyGen})$ to $\mathcal{S}$ and wait.
2. Upon receipt of $(\textbf{Key}, pk, \Pi, \pi)$ from $\mathcal{S}$, store (pk, Π, π), send $(\textbf{Key}, pk)$ to $\tilde{Q}$ and stop.

On message $(\textbf{Sign}, pk, m)$ from a user $\tilde{P}_i$:

1. Send $(\textbf{Sign}, P_i)$ to $\mathcal{S}$.
2. Upon receipt of $(\textbf{Sign}, P_i, \mathit{ack})$ from $\mathcal{S}$, send $(\textbf{Sign}, P_i?)$ to $\tilde{Q}$.
3. Upon receipt of $(\textbf{Sign}, P_i, \mathit{denied})$ from $\tilde{Q}$, send $(\textbf{Sign}, P_i, \mathit{denied})$ to $\mathcal{S}$. Wait for $(\textbf{Sign}, P_i, \mathit{denied}, \mathit{ack})$ from $\mathcal{S}$, and output $(\textbf{Sign}, \mathit{denied})$ to $\tilde{P}_i$.
4. Upon receipt of $(\textbf{Sign}, P_i)$ from $\tilde{Q}$, send $(\textbf{Sign}, P_i)$ to $\mathcal{S}$.
    1. Upon receipt of $(\textbf{Signature}, P_i, Q\ \mathit{completed})$ from $\mathcal{S}$, send $(\textbf{Signature}, P_i)$ to $\tilde{Q}$. If $\tilde{Q}$ is honest and $\tilde{P}_i$ is corrupt, increase the free signature count.
    2. Upon receipt of $(\textbf{Signature}, P_i, Q\ \mathit{not\ completed})$ from $\mathcal{S}$, send $(\textbf{Signature}, P_i, \mathit{not\ completed})$ to $\tilde{Q}$.
    3. Upon receipt of $(\textbf{Signature}, P_i, P_i\ \mathit{completed})$ from $\mathcal{S}$, if P_i is honest, $\sigma \overset{r}{\leftarrow} \Pi(m)$. If $(m, \sigma, 0)$ is stored, then stop. Otherwise, store $(m, \sigma, 1)$ and send $(\textbf{Signature}, \sigma)$ to $\tilde{P}_i$.
    4. Upon receipt of $(\textbf{Signature}, P_i, P_i\ \mathit{fail})$ from $\mathcal{S}$, send $(\textbf{Signature}, \mathit{fail})$ to $\tilde{P}_i$.

On message $(\textbf{Verify}, pk, m, \sigma)$ from an honest user $\tilde{P}_i$:

1. If $(m, \sigma, 1)$ is stored, send $(\textbf{Verify})$ to $\tilde{P}_i$ and stop.
2. If $(m, \sigma, 0)$ is stored, send $(\textbf{Verify}, \mathit{reject})$ to $\tilde{P}_i$ and stop.
3. If $\pi(m, \sigma) = 1$ and Q is corrupt, store $(m, \sigma, 1)$, send $(\textbf{Verify})$ to $\tilde{P}_i$ 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 $(\textbf{Verify})$ to $\tilde{P}_i$ and stop.
5. Store $(m, \sigma, 0)$ and send $(\textbf{Verify}, \mathit{reject})$ to $\tilde{P}_i$ and stop.

FIGURE 2. Blind signature functionality $\mathcal{F}_{\mathrm{BS}}$

The protocol $\pi_{\mathcal{BS}}$ 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 $\mathcal{F}_{\mathrm{BS}}$. We now prove our main result:

Theorem 2. *The blind signature scheme $\mathcal{BS}$ is secure with respect to weak blindness and non-forgeability if and only if the protocol $\pi_{\mathcal{BS}}$ securely realizes $\mathcal{F}_{BS}$ in the plain model.*

$\pi_{\mathcal{BS}}$ is defined as follows, with signer $Q(\tau)$ and users $P_1, \ldots, P_n$.

$Q(\tau)$:

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

P_i:

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

FIGURE 3. Blind signature protocol $\pi_{\mathcal{BS}}$

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

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

- Upon receiving $(\mathbf{KeyGen}, Q)$ from $\mathcal{F}_{\text{BS}}$, $\mathcal{S}$ runs $Gen(\tau)$, obtains a key pair (sk, pk) and stores (sk, pk). $\mathcal{S}$ then produces signature generation and verification facilities Π and π, and sends $(\mathbf{Key}, Q, pk, \Pi, \pi)$ to $\mathcal{F}_{\text{BS}}$.
- Upon receiving $(\mathbf{Sign}, P_i)$ from $\mathcal{F}_{\text{BS}}$, when both Q and P_i are honest: $\mathcal{S}$ simulates honest Q and honest P_i, and allows $\mathcal{A}$ to delay any communication between $\mathcal{S}|Q$ and $\mathcal{S}|P_i$.
    1. $\mathcal{S}$ chooses a random message $\tilde{m}$, and runs $User(pk, \tilde{m})$. When $User(pk, \tilde{m})$ outputs x on its communication tape, $\mathcal{S}|P_i$ sends (x) to $\mathcal{S}|Q$. When $\mathcal{S}|Q$, receives (x), $\mathcal{S}$ sends $(\mathbf{Sign}, P_i, ack)$ to $\mathcal{F}_{\text{BS}}$.
    2. Upon receipt of $(\mathbf{Sign}, P_i, denied)$ from $\mathcal{F}_{\text{BS}}$, $\mathcal{S}|Q$ sends $(\mathbf{Sign}, denied)$ to $\mathcal{S}|P_i$. When $\mathcal{S}|P_i$ receives $(\mathbf{Sign}, denied)$, $\mathcal{S}$ sends $(\mathbf{Sign}, P_i, denied, ack)$ to $\mathcal{F}_{\text{BS}}$.
    3. Upon receipt of $(\mathbf{Sign}, P_i)$ from $\mathcal{F}_{\text{BS}}$, $\mathcal{S}$ runs $Signer(sk, pk)$ and gives it x as input on its communication tape. $\mathcal{S}|Q$ forwards any output on *Signer*'s communication tape to $\mathcal{S}|P_i$, and vice versa. Similarly, $\mathcal{S}|P_i$ forwards any output on *User*'s communication tape to $\mathcal{S}|Q$, and vice versa.
    4. If *Signer* outputs *completed*, then $\mathcal{S}$ sends $(\mathbf{Signature}, P_i, Q\ completed)$ to $\mathcal{F}_{\text{BS}}$. Otherwise, if *Signer* outputs *not completed*, then $\mathcal{S}$ sends $(\mathbf{Signature}, P_i, Q\ not\ completed)$ to $\mathcal{F}_{\text{BS}}$.
    5. If *User* outputs σ, then $\mathcal{S}$ sends $(\mathbf{Signature}, P_i, P_i\ completed)$ to $\mathcal{F}_{\text{BS}}$. Otherwise, if *User* outputs *fail*, then $\mathcal{S}$ sends $(\mathbf{Signature}, P_i, P_i\ fail)$ to $\mathcal{F}_{\text{BS}}$.
- Upon receiving $(\mathbf{Sign}, P_i)$ from $\mathcal{F}_{\text{BS}}$, when Q is corrupt and P_i is honest: $\mathcal{S}$ simulates honest P_i.
    1. $\mathcal{S}$ chooses a random message $\tilde{m}$, and runs $User(pk, \tilde{m})$. When $User(pk, \tilde{m})$ outputs x on its communication tape, $\mathcal{S}|P_i$ sends (x) to $\mathcal{A}$.
    2. $\mathcal{S}$ sends $(\mathbf{Sign}, P_i, ack)$ to $\mathcal{F}_{\text{BS}}$, and $\mathcal{S}|\tilde{Q}$ receives $(\mathbf{Sign}, P_i?)$ from $\mathcal{F}_{\text{BS}}$.
    3. If $\mathcal{S}|P_i$ receives $(\mathbf{Sign}, denied)$ from $\mathcal{A}$, then $\mathcal{S}|\tilde{Q}$ sends $(\mathbf{Sign}, P_i, denied)$ to $\mathcal{F}_{\text{BS}}$. Upon receiving $(\mathbf{Sign}, P_i, denied)$ from $\mathcal{F}_{\text{BS}}$, $\mathcal{S}$ sends $(\mathbf{Sign}, P_i, denied, ack)$ to $\mathcal{F}_{\text{BS}}$.
    4. If $\mathcal{S}|P_i$ receives (y) from $\mathcal{A}$, then $\mathcal{S}|\tilde{Q}$ sends $(\mathbf{Sign}, P_i)$ to $\mathcal{F}_{\text{BS}}$, and $\mathcal{S}$ receives $(\mathbf{Sign}, P_i)$ from $\mathcal{F}_{\text{BS}}$. $\mathcal{S}|P_i$ forwards y to *User*'s communication tape. $\mathcal{S}|P_i$ forwards any output on *User*'s communication tape to $\mathcal{A}$, and vice versa.
    5. If *User* outputs σ, then $\mathcal{S}$ sends $(\mathbf{Signature}, P_i, P_i\ completed)$ to $\mathcal{F}_{\text{BS}}$. Otherwise, if *User* outputs *fail*, then $\mathcal{S}$ sends $(\mathbf{Signature}, P_i, P_i\ fail)$ to $\mathcal{F}_{\text{BS}}$.
    6. $\mathcal{S}$ sends $(\mathbf{Signature}, P_i, Q\ not\ completed)$ to $\mathcal{F}_{\text{BS}}$, and $\mathcal{S}|\tilde{Q}$ receives $(\mathbf{Signature}, P_i, Q\ not\ completed)$ from $\mathcal{F}_{\text{BS}}$.
- When Q is honest and P_i is corrupt: $\mathcal{S}$ simulates honest Q.

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

We want to show that if there exists an environment $\mathcal{Z}$ able to distinguish $\text{IDEAL}_{\mathcal{F}_{\text{BS}}}$ and $\pi_{\mathcal{BS}}$ with non-negligible probability, then we can construct an adversary A^{ror} breaking the real-or-random blindness or an adversary A^{nf} breaking the non-forgeability 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, $\mathcal{Z}$ interacts with the protocol $\pi_{\mathcal{BS}}$ 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 σ on a message m, P stores $(m, \sigma, 1)$. However, if P_i is corrupt and Q is honest, and Q outputs $(\textbf{Signature}, P_i)$, P increases the free signature count.

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

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

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

1. If $(m, \sigma, 1)$ is stored, $P|P_i$ outputs $(\textbf{Verify})$ and stops.
2. If $(m, \sigma, 0)$ is stored, $P|P_i$ outputs $(\textbf{Verify}, reject)$ and stops.
3. If $\pi(m, \sigma) = 1$ and Q is corrupt, P stores $(m, \sigma, 1)$, then $P|P_i$ outputs $(\textbf{Verify})$ 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 $(\textbf{Verify})$ and stops.
5. P stores $(m, \sigma, 0)$, then $P|P_i$ outputs $(\textbf{Verify}, reject)$ and stops.

We now prove the following lemma:

Lemma 1. *There is an adversary A^{nf} such that*

$$\Big| Pr[G_1 = 1] - Pr[G_2 = 1] \Big| \leq \boldsymbol{Succ}^{nf}_{\mathcal{BS},A^{nf}}(\tau).$$

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, σ), 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 σ 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

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

We now construct an adversary A^{nf} trying to break the non-forgeability of our scheme. $A^{\text{nf}}(1^\tau, pk)$ runs simulated copies of $\mathcal{Z}$, $\mathcal{A}$ 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 $(\textbf{Key}, pk)$. When Q engages in a protocol with some P_i, A^{nf} engages in a protocol with $\mathit{Signer}(sk, pk)$, forwarding inputs from P_i to $\mathit{Signer}(sk, pk)$ and vice versa. If F occurs, we deduce that for some numbers k, l such that $k > l$, $\mathcal{Z}$ has produced k valid pairs (m, σ), 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, σ). Hence we have

$$\mathbf{Succ}^{\text{nf}}_{\mathcal{BS},A^{\text{nf}}}(\tau) \geq \Pr[F],$$

which leads to

$$\Big| \Pr[G_1 = 1] - \Pr[G_2 = 1] \Big| \leq \mathbf{Succ}^{\text{nf}}_{\mathcal{BS},A^{\text{nf}}}(\tau),$$

by which the proof is complete. □

Game 3: In this game, $\mathcal{Z}$ interacts with the ideal protocol $\text{IDEAL}_{\mathcal{F}_{\text{BS}}}$ 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 $\mathit{User}(pk, m)$, while in Game 3, Q runs a protocol with $\mathit{User}(pk, \tilde{m})$, where $\tilde{m}$ is a randomly chosen message.

We prove the following lemma:

Lemma 2. *There is an adversary A^{ror} such that*

$$\Big| Pr[G_2 = 1] - Pr[G_3 = 1] \Big| = \boldsymbol{Adv}^{ror}_{\mathcal{BS},A^{ror}}(\tau).$$

Proof. We construct an adversary A^{ror} trying to break the real-or-random blindness of our scheme: $A^{\text{ror}}(1^\tau, sk, pk)$ runs simulated copies of $\mathcal{Z}$, $\mathcal{A}$ 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, A^{ror} simply outputs $(\mathbf{Key}, pk)$ in the name of Q. A^{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_i, A^{ror} outputs m_1 and engages in a protocol with $User(pk, m_b)$, where b is A^{ror}'s challenge bit and m_0 is a randomly chosen message. A^{ror} forwards messages from Q to $User(pk, m_b)$ and the other way around. If A^{ror} obtains σ as additional input, then A^{ror} outputs $(\mathbf{Signature}, \sigma)$ in the name of P_i. Finally, when $\mathcal{Z}$ outputs a bit b', A^{ror} outputs b'.

We observe that, if A^{ror}'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 A^{ror}'s advantage as follows:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ror}}_{\mathcal{BS},A^{\mathrm{ror}}}(\tau) &= \Big|\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]\Big| \\ &= \Big|\Pr[G_2 = 1] - \Pr[G_3 = 1]\Big|,\end{aligned}$$

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 A^{nf} or an effective A^{ror}. 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 A^{ror}, and construct an environment $\mathcal{Z}$ trying to distinguish interaction with $\pi_{\mathcal{BS}}$ from interaction with $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{BS}}}$. $\mathcal{Z}$ runs as follows:

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

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

$$\begin{aligned}
&\left|\Pr[\mathcal{Z}_{\pi_{\mathcal{BS}}} = 1] - \Pr[\mathcal{Z}_{\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{BS}}}} = 1]\right| \\
&= \left|\Pr[\mathcal{Z}_{\pi_{\mathcal{BS}}} = 1|b = 1] \cdot \Pr[b = 1] + \Pr[\mathcal{Z}_{\pi_{\mathcal{BS}}} = 1|b = 0] \cdot \Pr[b = 0] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2}\left(\Pr[\mathcal{Z}_{\pi_{\mathcal{BS}}} = 1|b = 1] + (1 - \Pr[\mathcal{Z}_{\pi_{\mathcal{BS}}} = 0|b = 0]) - 1\right)\right| \\
&= \frac{1}{2}\left|(\Pr[\mathcal{Z}_{\pi_{\mathcal{BS}}} = 1|b = 1] - \Pr[\mathcal{Z}_{\pi_{\mathcal{BS}}} = 0|b = 0])\right| \\
&= \frac{1}{2}\mathbf{Adv}^{\mathrm{ror}}_{\mathcal{BS},A^{\mathrm{ror}}}(\tau),
\end{aligned}$$

which completes this part of the proof.

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

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

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

If A^{nf} outputs $k > l$ valid pairs (m_i, σ_i), then if $\mathcal{Z}$ interacts with $\pi_{\mathcal{BS}}$, $\mathcal{Z}$'s output will be 1, while if $\mathcal{Z}$ interacts with $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{BS}}}$, $\mathcal{Z}$'s output will be 0. So we get

$$\begin{aligned}
\left|\Pr[\mathcal{Z}_{\pi_{\mathcal{BS}}} = 1] - \Pr[\mathcal{Z}_{\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{BS}}}} = 1]\right| &= \left|\Pr[k > l] - 0\right| \\
&= \mathbf{Succ}^{\mathrm{nf}}_{\mathcal{BS},A^{\mathrm{nf}}}(\tau),
\end{aligned}$$

by which the proof is complete. □

References

[1] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 394–403, Washington, DC, USA, 1997. IEEE Computer Society.

[2] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Available at `http://eprint.iacr.org/2000/067`.

[3] Ran Canetti and Marc Fischlin. Universally Composable Commitments. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 19–40, London, UK, 2001. Springer-Verlag.

[4] David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology-Crypto'82*, pages 199–203, 1982.

[5] Marc Fischlin. Round-Optimal Composable Blind Signatures in the Common Reference String Model. In *Advances in Cryptology-Crypto 2006*. Springer-Verlag, 2006.

[6] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of Blind Digital Signatures (Extended Abstract). In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 150–164, London, UK, 1997. Springer-Verlag.

[7] David Pointcheval and Jacques Stern. Provably Secure Blind Signature Schemes. In *ASIACRYPT '96: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 252–265, London, UK, 1996. Springer-Verlag.

Appendix A. Proof of Theorem 1

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:*
$\mathbf{Exp}^{lor}_{\mathcal{BS},A}(\tau)$*:*

1. $(sk, pk) \leftarrow Gen(\tau)$.
2. $b \leftarrow \{0, 1\}$.
3. *Run* $A(1^\tau, sk, pk)$.
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 τ) times.*
5. *A outputs a bit b'.*

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

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

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

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 $\Rightarrow$ left-or-right-blindness
(4) Left-or-right blindness $\Rightarrow$ weak blindness

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

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

(1) $j \leftarrow \{0,\ldots,n\}$, where n is the number of times that A^{lor} repeats step 4 in $\mathbf{Exp}^{\mathrm{lor}}_{\mathcal{BS},A^{\mathrm{lor}}}(\tau)$.
(2) Run A^{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^{lor} during the run. When A^{lor} outputs (m_{0i}, m_{1i}):
   - If $0 \le i < j$, let A^{lor} engage in two parallel interactive protocols, the first with $\mathit{User}(pk, m_{0i})$ and the second with $\mathit{User}(pk, m_{1i})$. If $\mathit{User}(pk, m_{0i})$ outputs $\sigma(m_{0i})$ and $\mathit{User}(pk, m_{1i})$ outputs $\sigma(m_{1i})$, then A^{lor} is given $(\sigma(m_{0i}), \sigma(m_{1i}))$ as additional input.
   - If $i = j$, then output (m_{0i}, m_{1i}). Forward inputs from $\mathit{User}(pk, m_{bi})$ and $\mathit{User}(pk, m_{(1-b)i})$ to the respective communication tapes of A^{lor}, and the other way around. If $(\sigma(m_{0i}), \sigma(m_{1i}))$ is obtained as additional input, forward this to A^{lor}.
   - If $j < i < n$, let A^{lor} engage in two parallel interactive protocols, the first with $\mathit{User}(pk, m_{1i})$ and the second with $\mathit{User}(pk, m_{0i})$. If $\mathit{User}(pk, m_{0i})$ outputs $\sigma(m_{0i})$ and $\mathit{User}(pk, m_{1i})$ outputs $\sigma(m_{1i})$, then A^{lor} is given $(\sigma(m_{0i}), \sigma(m_{1i}))$ as additional input.

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

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

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

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

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

$$\Big| (P_0 - P_1) + (P_1 - P_2) + \cdots + (P_{n-1} - P_n) \Big| = \delta(\tau).$$

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^{ror} outputs 1 is P_j. Hence we get

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{wb}}_{\mathcal{BS},A^{\mathrm{wb}}}(\tau) &= \Big|\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]\Big| \\
&= \Big|\Pr[b' = 1|b = 1 \wedge j = 0] \cdot \Pr[j = 0] + \cdots + \\
&\quad \Pr[b' = 1|b = 1 \wedge j = n-1] \cdot \Pr[j = n-1] \\
&\quad -\Pr[b' = 1|b = 0 \wedge j = 0] \cdot \Pr[j = 0] - \cdots - \\
&\quad \Pr[b' = 1|b = 0 \wedge j = n-1] \cdot \Pr[j = n-1]\Big| \\
&= \left|\frac{1}{n}\left(P_0 + \cdots + P_{n-1} - (P_1 + \cdots + P_n)\right)\right| \\
&= \frac{1}{n}\Big|(P_0 - P_1) + \cdots + (P_{n-1} - P_n)\Big| \\
&= \frac{\delta(\tau)}{n}.
\end{aligned}$$

Since $\delta(\tau)$ is non-negligible in τ, and n is polynomial in τ, $\frac{1}{n}\delta(\tau)$ is also non-negligible in τ.

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

Algorithm $A^{\mathrm{lor}}(1^\tau, sk, pk)$:
(1) Run A^{ror} on input $(1^\tau, sk, pk)$.
(2) Each time A^{ror} outputs a message m_1:
- Choose a random message m_0 and output (m_0, m_1).
- Forward inputs from $User(pk, m_b)$ to A^{ror}'s communication tape, and the other way around. Run the protocol with $User(pk, m_{1-b})$ 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^{ror}.

(3) When A^{ror} outputs a bit b', output b'.

We observe that when A^{lor}'s challenge bit $b = 0$, A^{lor} simulates A^{ror}'s environment in $\mathbf{Exp}^{\mathrm{ror}}_{\mathcal{BS},A^{\mathrm{ror}}}(\tau)$ in the case where A^{ror}'s challenge bit $b = 0$, and correspondingly for the case $b = 1$. Hence we get

$$\mathbf{Adv}^{\mathrm{lor}}_{\mathcal{BS},A^{\mathrm{lor}}}(\tau) = \mathbf{Adv}^{\mathrm{ror}}_{\mathcal{BS},A^{\mathrm{ror}}}(\tau),$$

which, by assumption, is non-negligible in τ.

Real-or-Random Blindness $\Rightarrow$ 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^{ror} 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 $\mathbf{Exp}^{\text{lor}}_{\mathcal{BS},A}(\tau)$, 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 $User(pk, m_0)$ and $User(pk, m_1)$, respectively. Moreover, we run internally a protocol between honest $Signer(sk, pk)$ and honest $User(pk, m_1')$, and obtain $\sigma(m_1')$. If $User(pk, m_0)$ outputs $\sigma(m_0)$ and $User(pk, m_1)$ 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 $User(pk, m_0)$ and $User(pk, m_1)$, respectively. Moreover, we run two protocols internally, one between honest $Signer(sk, pk)$ and honest $User(pk, m_0')$, the other between honest $Signer(sk, pk)$ and honest $User(pk, m_1')$. We thus obtain $\sigma(m_0')$ and $\sigma(m_1')$. If $User(pk, m_0)$ outputs $\sigma(m_0)$ and $User(pk, m_1)$ 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^{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^{lor} on input $(1^\tau, sk, pk)$.
(2) $\tilde{b} \leftarrow \{0, 1\}$
(3) Each time A^{lor} outputs a message pair, say $(\tilde{m}_0, \tilde{m}_1)$:
   - Output $(\tilde{m}_{\tilde{b}}, \tilde{m}_{1-\tilde{b}})$.
   - Engage in two protocols, say with $User_0$ and $User_1$, respectively. Forward outputs from $User_0$ and $User_1$ to the respective communication tapes of A^{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^{lor}.
(4) When A^{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^{lor}'s environment in $\mathbf{Exp}^{\text{lor}}_{\mathcal{BS},A^{\text{lor}}}(\tau)$, with $\tilde{b}$ acting as the challenge bit. On the other hand, if A^{13} participates in Game 3, A^{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}:

$$\mathbf{Adv}^{13}_{\mathcal{BS},A}(\tau) = \Big|\Pr[b'' = 1|\text{Game 1}] - \Pr[b'' = 1|\text{Game 3}]\Big|,$$

where

$$\begin{aligned}\Pr[b''=1|\text{Game }1] &= \Pr[b'=1|\text{Game }1\wedge\tilde{b}=1]\cdot\Pr[\tilde{b}=1]\\ &\quad+\Pr[b'=0|\text{Game }1\wedge\tilde{b}=0]\cdot\Pr[\tilde{b}=0]\\ &= \Pr[b'=1|\text{Game }1\wedge\tilde{b}=1]\cdot\frac{1}{2}\\ &\quad+\Big(1-\Pr[b'=1|\text{Game }1\wedge\tilde{b}=0]\Big)\cdot\frac{1}{2}.\end{aligned}$$

So we get

$$\begin{aligned}\mathbf{Adv}^{13}_{\mathcal{BS},A}(\tau) &= \left|\frac{1}{2}\Big(\Pr[b'=1|\text{Game }1\wedge\tilde{b}=1]-\Pr[b'=1|\text{Game }1\wedge\tilde{b}=0]\Big)+\frac{1}{2}-\frac{1}{2}\right|\\ &= \frac{1}{2}\cdot\mathbf{Adv}^{\text{lor}}_{\mathcal{BS},A}(\tau),\end{aligned}$$

which, by assumption, is non-negligible in τ. 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^{ror} with respect to real-or-random blindness.

First, we assume that there exists an effective A^{12} and consider the experiment $\mathbf{Exp}^{\text{ror}}_{\mathcal{BS},A^{\text{ror}}}(\tau)$, where A^{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 $User(pk,\tilde{m}_0)$ internally, and engage in a protocol, say with $User_1$. Forward inputs from $User(pk,\tilde{m}_0)$ and $User_1$ to the respective communication tapes of A^{12}, and the other way around. If $User(pk,\tilde{m}_0)$ 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^{ror}'s challenge bit $b=0$, then A^{ror} simulates A^{12}'s environment in Game 2, while if $b=1$, A^{ror} simulates A^{12}'s environment in Game 1. We compute the advantage of A^{ror} as follows:

$$\begin{aligned}\mathbf{Adv}^{\text{ror}}_{\mathcal{BS},A}(\tau) &= \Big|\Pr[b'=1|b=1]-\Pr[b'=1|b=0]\Big|\\ &= \Big|\Pr[b'=1|\text{Game }1]-\Pr[b'=1|\text{Game }2]\Big|\\ &= \mathbf{Adv}^{12}_{\mathcal{BS},A}(\tau),\end{aligned}$$

which, by assumption, is non-negligible in τ.

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

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

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

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ror}}_{\mathcal{BS},A}(\tau) &= \Big|\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]\Big| \\ &= \Big|\Pr[b' = 1|\text{Game 2}] - \Pr[b' = 1|\text{Game 3}]\Big| \\ &= \mathbf{Adv}^{23}_{\mathcal{BS},A}(\tau),\end{aligned}$$

which, by assumption, is non-negligible in τ.

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 $\mathbf{Exp}^{\mathrm{lor}}_{\mathcal{BS},A^{\mathrm{lor}}}(\tau)$, where A^{lor} proceeds as described below.

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

(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 $\mathbf{Exp}^{\mathrm{wb}}_{\mathcal{BS},A^{\mathrm{wb}}}(\tau)$ in the case where A^{wb}'s challenge bit $b = 0$, and correspondingly for the case where $b = 1$. Hence we get

$$\mathbf{Adv}^{\mathrm{lor}}_{\mathcal{BS},A^{\mathrm{lor}}}(\tau) = \mathbf{Adv}^{\mathrm{wb}}_{\mathcal{BS},A^{\mathrm{wb}}}(\tau),$$

which, by assumption, is non-negligible in τ.

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-forgeability*. 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 $\mathbf{G}$ be a multiplicative cyclic group of prime order p, and let g be a generator of $\mathbf{G}$. We say that $\mathbf{G}$ is a *bilinear group* if the group operation in $\mathbf{G}$ is efficiently computable, and if there exists a multiplicative cyclic group $\mathbf{G}_1$ of order p and an efficiently computable non-degenerate bilinear map $e : \mathbf{G} \times \mathbf{G} \to \mathbf{G}_1$, i.e. for all $u, v \in \mathbf{G}$ 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 $\mathcal{S}$, and note that we use the following notation: $\mathcal{S} = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$, where $\mathit{Gen}(1^\tau)$ outputs (sk, pk), $\mathit{Sign}(sk, m)$ outputs σ, and $\mathit{Verify}(pk, m, \sigma)$ 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 $\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{S},A}(\tau)$, given in Figure 1. The experiment $\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{S},A}(\tau)$ 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, σ) such that σ is a valid signature on m with respect to pk. A has access to the signing oracle $\mathcal{O}_{\mathcal{S}}$, which takes a message as input and outputs a signature on the message under sk. It is required that $\mathcal{O}_{\mathcal{S}}$ 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 $\mathcal{S}$ with respect to UF-CMA as

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

Definition 1. *The scheme $\mathcal{S}$ is said to be (t, q, ϵ)-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 ϵ.*

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 $\mathbf{G}$ be a bilinear group of order p, where p has length τ, and let $e : \mathbf{G} \times \mathbf{G} \to \mathbf{G}_1$ be the corresponding efficiently computable bilinear map. To generate the public key, the algorithm Gen_W chooses a random generator $g \in \mathbf{G}$ and a random $\alpha \in \{0, \ldots, p-1\}$ and lets $g_1 = g^\alpha$. Additionally, for some appropriate n, it chooses random $g_2, u', u_1, \ldots, u_n \in \mathbf{G}$ 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^α.

Signature Generation: Upon input of a message m of length n, the algorithm $Sign_W$ chooses a random $r \in \{0, \dots, p-1\}$ and computes the signature σ as $\sigma = (\sigma_1, \sigma_2) = (g_2^\alpha \left(u' \prod_{i=1}^n u_i^{m_i}\right)^r, g^r)$, 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 $Verify_W$ 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 (σ_1, σ_2) on a message m is generated according to the above scheme. Note that, if we randomly choose $r^* \in \{0, \dots, p-1\}$ and let $(\sigma_1^*, \sigma_2^*) = (\sigma_1 (u' \prod_{i=1}^n u_i^{m_i})^{r^*}, \sigma_2 g^{r^*})$, (σ_1^*, σ_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, ϵ)-secure with respect to UF-CMA if the $(t, \frac{\epsilon}{16(n+1)q})$-CDH assumption holds in $\mathbf{G}$.

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: $\mathcal{PKE} = (Gen, Enc, Dec)$, where $Gen(1^\tau)$ outputs (sk, pk), $Enc(pk, m)$ outputs c, and $Dec(sk, c)$ outputs $m/\perp$.

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 $\mathbf{Exp}^{\text{ror-cpa}}_{\mathcal{PKE},A}(\tau)$ given in Figure 1. In the experiment $\mathbf{Exp}^{\text{ror-cpa}}_{\mathcal{PKE},A}(\tau)$, the adversary A has access to the oracle $\mathcal{O}^b_{\text{ror}}$ (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 pk. A new random string is chosen for each query. If $b = 1$, it outputs an encryption of m under 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

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

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

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, Gen_L outputs (sk_L, pk_L), where pk_L is a triple of randomly chosen generators $\alpha_1, \alpha_2, \beta \in \mathbf{G}$, where $\mathbf{G}$ is a group of prime order p, and p has length τ. sk_L is the exponents $a_1, a_2 \in \{0, \dots, p-1\}$ such that $\alpha_1^{a_1} = \alpha_2^{a_2} = \beta$. Enc_L takes as input a message $m \in \mathbf{G}$, chooses random values $r, s \in \{0, \dots, p-1\}$, and outputs the triple $(y_1, y_2, y_3) = (\alpha_1^r, \alpha_2^s, \beta^{r+s} m)$. Dec_L takes a ciphertext (y_1, y_2, y_3) as input and outputs $y_3 y_1^{-a_1} y_2^{-a_2}$.

```
$\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{S},A}(\tau)$:
    1. $(sk, pk) \leftarrow Gen(1^\tau)$.
    2. $(m, \sigma) \leftarrow A^{\mathcal{O}_S}(pk)$.
    3. If $Verify(pk, m, \sigma) = accept$
       then return 1, otherwise
       return 0.
$\mathbf{Exp}^{\text{nf}}_{\mathcal{BS},A}(\tau)$:
    1. $crs \leftarrow D(1^\tau)$.
    2. $(pk_{KS}, sk_{KS}) \leftarrow KS(1^\tau)$.
    3. $(pk, sk) \leftarrow Gen(1^\tau)$.
    4. Let $A(1^\tau, crs, pk_{KS}, pk)$
       engage in polynomially many
       (in $\tau$) parallel interactive
       protocols, with polynomially
       many (in $\tau$) copies of
       $Signer(pk, sk)$, where $A$
       decides in an adaptive manner
       when to stop. Let $l$ be the
       number of executions, where
       $Signer$ outputs $completed$.
    5. $A$ outputs a collection
       $\{(m_1, \sigma(m_1)), \ldots, (m_k, \sigma(m_k))\}$,
       subject to the constraint that
       $m_i \neq m_j$ for $1 \leq i < j \leq k$,
       and $Verify(pk, m_i, \sigma(m_i))$
       outputs $accept$ for $1 \leq i \leq k$.
```

```
$\mathbf{Exp}^{\text{ror-cpa}}_{\mathcal{PKE},A}(\tau)$
    1. $(sk, pk) \leftarrow Gen(1^\tau)$
    2. $b \leftarrow \{0, 1\}$
    3. $b' \leftarrow A^{\mathcal{O}^b_{\text{ror}}}(pk)$
    4. Return $b'$.
$\mathbf{Exp}^{\text{b}}_{\mathcal{BS},A}(\tau)$:
    1. $crs \leftarrow D(1^\tau)$.
    2. Run $A$ on input $(1^\tau, crs)$.
    3. $(pk, r, (m_0, m_1)) \leftarrow A$.
    4. $(sk_{KS}, pk_{KS}) \leftarrow KS(1^\tau, r)$.
    5. $b \leftarrow \{0, 1\}$.
    6. Let $A$ engage in two parallel
       interactive protocols, the first
       with $User(pk, m_b)$ and the
       second with $User(pk, m_{1-b})$.
    7. 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.
    8. $A$ outputs a bit $b'$.
```

FIGURE 1. Experiments for security definitions.

It can be shown that, for all t, q polynomial in τ, the LE scheme is (t, q, ϵ)-secure with respect to ROR-CPA, for some ϵ negligible in τ, if the corresponding holds for the DLDH problem in $\mathbf{G}$.

2.4. **Setup Assumptions.**

The Common Reference String Model: Let D be a probabilistic polynomial-time algorithm which takes a security parameter 1^τ 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 $\mathcal{F}^D_{\text{crs}}$, 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^τ 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 $\mathcal{F}_{\mathrm{reg}}^{KS}$, 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 $\mathcal{F}_{\mathrm{reg}}^{KS}$ the random coins r used to create his key pair. $\mathcal{F}_{\mathrm{reg}}^{KS}$ 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 Σ-Protocols in the Registered Public Key Model.

According to [17], a *Σ-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 Σ-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 Σ-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 Σ-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 $\mathcal{F}_{\mathrm{reg}}^{KS}$ 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 $\mathcal{F}_{\mathrm{reg}}^{KS}$, and computes the first message a according to the Σ-protocol. Then, P exploits the homomorphic property of the encryption scheme, and the fact that the Σ-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 Σ-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 Σ-protocol. Furthermore, since a simulator running V obtains the random coins intended for $\mathcal{F}_{\mathrm{reg}}^{KS}$, 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 Σ-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 Σ-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 $\mathcal{F}_{\mathrm{crs}}^{D}$ and $\mathcal{F}_{\mathrm{reg}}^{KS}$ defined earlier, and where the signer is initially required to register a public key pk_{KS} with $\mathcal{F}_{\mathrm{reg}}^{KS}$.

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

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

It is required that for any message m, and for all key pairs (sk, pk) output by Gen, if both Signer and User follow the protocol, then Signer(sk, pk) outputs completed, User(pk, m) outputs $\sigma(m)$, and 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-forgeability for blind signature schemes in our model is formally defined using the experiment $\mathbf{Exp}^{\mathrm{nf}}_{\mathcal{BS},A}(\tau)$ 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 $\mathcal{BS}$ with respect to non-forgeability as

$$\mathbf{Succ}^{\mathrm{nf}}_{\mathcal{BS},A} = \Pr\left[k > l\right].$$

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

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

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

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

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 π that c was honestly generated. The signer responds with σ', which is essentially a signature on the commitment. In the final step, the user obtains a blind signature σ 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 π. Let A^{nf} be an adversary trying to break the non-forgeability of our scheme. In order to achieve provable security, π is constructed such that a simulator running a copy of A^{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 (σ_1', σ_2'), hence the non-forgeability of our scheme reduces to the UF-CMA security of Waters' scheme.

The proof π should convince the verifier (in this case the signer) that c was honestly generated. If we let $t_{\tau-1} \ldots t_o$ 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, $0 \le i \le \tau - 1$, and m_i, $1 \le i \le n$. This is obtained by having the prover (in this case *User*) commit to each of the values g^{t_i}, $0 \le i \le \tau - 1$, and $u_i^{m_i}$,

Signer		*User*
$pk_{BS} = (g, g_1, g_2, u', U)$		$pk_{BS} = (g, g_1, g_2, u', U)$
$sk_{BS} = g_2^{\alpha}$		$m = m_1 m_2 \dots m_n \in \{0,1\}^n$
		Choose $t \leftarrow \{0, \dots, p-1\}$
		Let $c = g^t \prod_{i=1}^{n} u_i^{m_i}$
		Check that $u'c \neq 1$
		Compute a proof π of correctness of c
	$\xleftarrow{(c,\pi)}$	
Verify π		
Choose $r \leftarrow \{0, \dots, p-1\}$		
Let $\sigma_1' = g_2^{\alpha}(u'c)^r$		
Let $\sigma_2' = g^r$		
Output		
completed/*not completed*	$\xrightarrow{(\sigma_1',\sigma_2')}$	
		Choose $r' \leftarrow \{0, \dots, p-1\}$
		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 (σ_1, σ_2)/*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 $Enc_L(m, r, s)$ 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, $Enc_L(m, r, s) = (\alpha_1^r, \alpha_2^s, \beta^{r+s} m)$, and note that this encryption scheme is homomorphic, i.e.

$$Enc_L(m, r, s) \cdot Enc_L(m', r', s') = Enc_L(mm', r + r', s + s).$$

Let $t_{l-1} \dots t_o$ 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 $r_i, s_i \in \{0, \dots, p-1\}$

and computing

$$T_i = Enc_L(g^{t_i}, r_i, s_i) = (\alpha_1^{r_i}, \alpha_2^{s_i}, \beta^{r_i+s_i} g^{t_i}).$$

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

$$M_i = Enc_L(u_i^{m_i}, r'_i, s'_i) = (\alpha_1^{r'_i}, \alpha_2^{s'_i}, \beta^{r'_i+s'_i} u_i^{m_i}).$$

Correctness of each of the above commitments can be proved using a suitable Σ-protocol. Moreover, since we want our proof π to be non-interactive, we apply the technique developed by Damgård et al. for compiling Σ-protocols into NIZK proofs.

As for proving that c in fact contains the committed values, note that, by letting $r^* = \sum_{i=0}^{l-1} r_i 2^i + \sum_{i=1}^{n} r'_i$ and $s^* = \sum_{i=0}^{l-1} s_i 2^i + \sum_{i=1}^{n} s'_i$, we have

$$\prod_{i=0}^{l-1} T_i^{2^i} \prod_{i=1}^{n} M_i = (\alpha_1^{r^*}, \alpha_2^{s^*}, \beta^{r^*+s^*} g^t \prod_{i=1}^{n} u_i^{m_i}),$$

i.e. $\prod_{i=0}^{l-1} T_i^{2^i} \prod_{i=1}^{n} M_i$ is a commitment to $g^t \prod_{i=1}^{n} u_i^{m_i}$. This means that the prover can prove the correctness of c by opening this commitment, that is, by including r^* and s^* in π. 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

$$Enc_L(c, r^*, s^*) = \prod_{i=0}^{l-i} T_i^{2^i} \prod_{i=1}^{n} M_i.$$

4.2. **The Protocol** *compile*(Σ_{OR})**.** The proof π 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 α_1, α_2, β and u are publicly known elements of a bilinear group $\mathbf{G}$ 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 Σ-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 Σ-protocols Σ_l and Σ_r for relations R_l and R_r, the OR-construction yields a Σ-protocol Σ_{OR} for the relation R_{OR} defined by

$$((x_l, x_r), (w_l, w_r)) \in R_{OR} \Leftrightarrow (x_l, w_l) \in R_l \vee (x_r, w_r) \in R_r.$$

Let R_L be the relation defined by

$$(x, w) = ((x_1, x_2, x_3), (r, s)) \in R_L \Leftrightarrow x_1 = \alpha_1^r, x_2 = \alpha_2^s, x_3 = \beta^{r+s}.$$

We note that, for a commitment (y_1, y_2, y_3), we have

$$(y_1, y_2, y_3) = (\alpha_1^r, \alpha_2^s, \beta^{r+s}) \Leftrightarrow ((y_1, y_2, y_3), (r, s)) \in R_L$$

and

$$(y_1, y_2, y_3) = (\alpha_1^r, \alpha_2^s, \beta^{r+s} u) \Leftrightarrow ((y_1, y_2, \frac{y_3}{u}), (r, s)) \in R_L.$$

This means that a suitable protocol Σ_{OR} for our purpose is obtained by composing Σ-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 Σ-protocol Σ_L for R_L is presented in Appendix B, where we also show that Σ_L has completeness, relaxed special soundness and perfect honest-verifier zero-knowledge.

The protocol Σ_{OR} is constructed in a standard way from the two suitable instances of Σ_L, and it is not hard to show that completeness, relaxed special soundness and perfect honest-verifier zero-knowledge of Σ_{OR} follow from the corresponding properties of Σ_L. For completeness, a description of Σ_{OR} and proofs of the respective properties are given in Appendix C. We simply note that Σ_{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 Σ_{OR} has linear answer, hence we may apply the compilation technique of Damgård et al.

We now describe the compilation of the protocol Σ_{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 $\mathcal{F}_{\mathrm{reg}}^{KS}$, 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 2τ as input. Choose a random challenge $e \in \{0, \ldots, p-1\}$. Also, let c be a Paillier encryption of e under pk_P. The public key is (pk_P, c) and the private key is (sk_P, e).

Due to the homomorphic property of Paillier encryption, and the fact that Σ_{OR} has linear answer, it is possible to execute the prover's side of the protocol given only the encryption c 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 $Enc_P(pk_P, u_i) \cdot c^{v_i}$.

The compiled protocol is defined below. We remind the reader that Σ_{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 *Compile*(Σ_{OR}):

(1) Given (x, w), P gets V's public key (pk_P, c) from $\mathcal{F}_{\mathrm{reg}}^{KS}$ and computes the first message a according to Σ_{OR}. Then, for i such that $1 \leq i \leq 6$, P computes $Enc_P(pk_P, u_i) \cdot c^{v_i}$, and lets c_i be a randomization of the resulting encryption. P sends (x, π) 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 be the Paillier decryption of c_i under sk_P. Then V verifies that the conversation $(a, e, (z'_1, \ldots, z'_6))$ would be accepted by the verifier of Σ_{OR} upon input x, and accepts or rejects accordingly.

We need a result from [17] based on the following assumption: '$\mathcal{H}_{\mathrm{Paillier}}$ is 2-harder than $\mathcal{G}_{\mathrm{dlog}}$'. 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 τ, there is no algorithm that breaks Paillier encryption for moduli of length 2τ, with runtime comparable to that of A. We note that, as defined in [17], $\mathcal{G}_{\text{dlog}}$ addresses the DLOG problem in the subgroup of $\mathbf{Z}_p$ of order p', where p and p' are primes, and $p = 2p' + 1$. Since our Σ-protocol involves a bilinear group, we need a modified version of $\mathcal{G}_{\text{dlog}}$, addressing the DLOG problem in a general bilinear group of prime order. We call this modified version $\mathcal{G}^*_{\text{dlog}}$. By arguing as in [17], the following assumption seems reasonable.

Assumption 6. *$\mathcal{H}_{Paillier}$ is 2-harder than $\mathcal{G}^*_{dlog}$.*

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(Σ_{OR}) has correctness and perfect zero-knowledge (in the registered public-key model). Furthermore, under Assumption 6, compile(Σ_{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 $\mathcal{F}^D_{\text{crs}}$ and $\mathcal{F}^{KS}_{\text{reg}}$ defined earlier. Furthermore, *Signer* is initially required to register a public key pk_{KS} with the functionality $\mathcal{F}^{KS}_{\text{reg}}$.

Common Reference String: The algorithm D takes 1^τ as input, randomly chooses $\alpha_1, \alpha_2, \beta \in \mathbf{G}$ and lets $crs = \alpha_1||\alpha_2||\beta$.

Key Setup: The algorithm *KS* works exactly as in *compile*(Σ_{OR}), i.e. the public and secret keys are $pk_{KS} = (pk_P, c)$ and $sk_{KS} = (sk_P, e)$.

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 $sk_{BS} = g_2^\alpha$.

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 π of correctness of c as follows: Let $t_{\tau-1} \ldots t_o$ be the bit representation of t, that is, $t = \sum_{i=0}^{\tau-1} t_i 2^i$. *User* gets $crs = \alpha_1||\alpha_2||\beta$ from $\mathcal{F}^D_{\text{crs}}$ and lets $pk_L = (\alpha_1, \alpha_2, \beta)$. Then, for i, $0 \leq i \leq \tau - 1$, he chooses random values $r_i, s_i \in \{0, \ldots, p-1\}$ and computes

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

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

$$r^* = \sum_{i=0}^{\tau-1} r_i 2^i + \sum_{i=1}^{n} r'_i, \;\; s^* = \sum_{i=0}^{\tau-1} s_i 2^i + \sum_{i=1}^{n} s'_i.$$

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

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

$$Enc_L(pk_L, c, r^*, s^*) = \prod_{i=0}^{l-i} T_i^{2^i} \prod_{i=1}^{n} M_i.$$

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 $\sigma'_1 = g_2^{\alpha}(u'c)^r$ and $\sigma'_2 = g^r$, outputs *completed*, sends (σ'_1, σ'_2) to *User* and stops.

Upon receiving (σ'_1, σ'_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 (σ_1, σ_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 $\mathcal{O}(\log \tau)$ 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*(Σ_{OR}). Hence, under this assumption, we may achieve non-forgeability 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', ϵ')-secure with respect to UF-CMA, then our blind signature scheme is (t, l, ϵ)-secure with respect to non-forgeability, where $q' = l = \mathcal{O}(\log \tau)$, $t' = t + poly(\tau)$ and $\epsilon' = (1 - \rho)\epsilon$, where ρ is a negligible function in τ.*

Theorem 9. *If the LE scheme is $(t', 1, \epsilon')$-secure with respect to ROR-CPA, then our blind signature scheme is (t, ϵ)-secure with respect to blindness, where $t' = t + poly(\tau)$, and $\epsilon' = \frac{\epsilon}{2n+2\tau}$.*

References

[1] Masayuki Abe. A Secure Three-Move Blind Signature Scheme for Polynomially Many Signatures. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 136–151, London, UK, 2001. Springer-Verlag.

[2] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally Composable Protocols with Relaxed Set-Up Assumptions. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 186–195, Washington, DC, USA, 2004. IEEE Computer Society.

[3] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 26–45, London, UK, 1998. Springer-Verlag.

[4] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The Power of RSA Inversion Oracles and the Security of Chaum's RSA-Based Blind Signature Scheme. In *FC '01: Proceedings of the 5th International Conference on Financial Cryptography*, pages 319–338, London, UK, 2002. Springer-Verlag.

[5] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public-Key Cryptography (PKC) 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer-Verlag, 2003.

[6] D. Boneh, X. Boyen, and H. Shacham. Short Group Signatures. pages 41–55, 2004.

[7] Jan Camenisch, Maciej Koprowski, and Bogdan Warinschi. Efficient Blind Signatures Without Random Oracles. In *SCN*, pages 134–148, 2004.

[8] David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology-Crypto'82*, pages 199–203, 1982.

[9] Marc Fischlin. Round-Optimal Composable Blind Signatures in the Common Reference String Model. In *Advances in Cryptology-Crypto 2006*. Springer-Verlag, 2006.

[10] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[11] Carmit Hazay, Jonathan Katz, Chiu-Yuen Koo, and Yehuda Lindell. Concurrently-Secure Blind Signatures Without Random Oracles or Setup Assumptions. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007.

[12] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of Blind Digital Signatures (Extended Abstract). In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 150–164, London, UK, 1997. Springer-Verlag.

[13] Aggelos Kiayias and Hong-Sheng Zhou. Concurrent Blind Signatures Without Random Oracles. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2006.

[14] Tatsuaki Okamoto. Efficient Blind and Partially Blind Signatures Without Random Oracles. In *TCC*, pages 80–99, 2006.

[15] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - EUROCRYPT '99*.

[16] David Pointcheval and Jacques Stern. Provably Secure Blind Signature Schemes. In *ASIACRYPT '96: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 252–265, London, UK, 1996. Springer-Verlag.

[17] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-Interactive Zero-Knowledge from Homomorphic Encryption. In *TCC*, pages 41–59, 2006.

[18] Brent Waters. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*, pages 114–127, 2005.

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 $\mathbf{G}$ of order p and a generator $g \in \mathbf{G}$, the challenger chooses random $a, b \in \{0, \dots, p-1\}$ and outputs (g, g^a, g^b). The adversary's job is to find $g^{ab} \in \mathbf{G}$.

We define the *success rate* of an adversary A in solving the CDH problem in $\mathbf{G}$ as

$$\mathbf{Succ}_A^{\mathrm{CDH}} = \Pr\left[A(g, g^a, g^b) = g^{ab}\right].$$

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

A.2. **The Decision Linear Diffie-Hellman Assumption.** The *Decision Linear Diffie-Hellman (DLDH) problem* is defined as follows: Given a cyclic group $\mathbf{G}$ of order p and generators $g_1, g_2, g_3 \in \mathbf{G}$, the challenger chooses random $a, b, c \in \{0, \dots, p-1\}$ and a random bit β. 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 β' of β.

We say that an adversary A has an *advantage* at least ϵ in solving the DLDH problem in $\mathbf{G}$ if

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

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

Appendix B. The Protocol Σ_L

The Σ-protocol Σ_L for the relation R_L is presented in Figure 3.

Σ_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

$$\alpha_1^{z_1} = x_1^e a_1, \ \alpha_1^{z_1'} = x_1^{e'} a_1,$$

which implies that

$$\alpha_1^{z_1 - z_1'} = x_1^{e-e'}.$$

Assuming that

$$x_1 = \alpha_1^r,$$

P		V
(x_1, x_2, x_3)		(x_1, x_2, x_3)
(r, s)		
Choose $r', s' \leftarrow \{0, \dots, p-1\}$		
Let		
$a_1 = \alpha_1^{r'}$		
$a_2 = \alpha_2^{s'}$		
$a_3 = \beta^{r'+s'}$		
	$\xrightarrow{(a_1,a_2,a_3)}$	
		Choose $e \leftarrow \{0, \dots, p-1\}$
	$\xleftarrow{e}$	
Let		
$z_1 = r' + rc$		
$z_2 = s' + sc$		
	$\xrightarrow{(z_1,z_2)}$	
		Check if
		$\alpha_1^{z_1} = x_1^e a_1$
		$\alpha_2^{z_2} = x_2^e a_2$
		$\beta^{z_1+z_2} = x_3^e a_3$
		and accept or reject accordingly

FIGURE 3. The protocol Σ_L.

r can be computed as

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

We also have

$$\alpha_2^{z_2} = x_2^e a_2,\ \alpha_2^{z_2'} = x_2^{e'} a_2,$$

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

$$\beta^{z_1+z_2} = x_3^e a_3, \ \beta^{z_1'+z_2'} = x_3^{e'} a_3,$$

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

Assuming that $(x_1, x_2, x_3) = (\alpha_1^r, \alpha_2^s, \beta^{r+s})$, we have

$$a_1^* = \alpha_1^{z_1^*-re}, \ a_2^* = \alpha_2^{z_2^*-se}, \ a_3^* = \beta^{(z_1^*-re)+(z_2^*-se)}$$

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 Σ_L has perfect honest-verifier zero-knowledge.

Appendix C. The Protocol Σ_{OR}

In this section we describe the protocol Σ_{OR} constructed from two instances of the protocol Σ_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 $((y_1, y_2, \frac{y_3}{u}), (r, s)) \in R_L$, without revealing which is the case. That is, a suitable relation R_{OR} can be defined by

$$(x, w) = ((y_1, y_2, y_3), (r, s)) \in R_{OR} \Leftrightarrow ((y_1, y_2, y_3), (r, s)) \in R_L \vee ((y_1, y_2, \frac{y_3}{u}), (r, s)) \in R_L.$$

We note that, if $(y_1, y_2, y_3) \in L_{R_L}$, then we have $(y_1, y_2, \frac{y_3}{u}) \notin L_{R_L}$ 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 $(y_1, y_2, \frac{y_3}{u})$ and produces accepting conversations for both statements.

The protocol Σ_{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 Σ_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 $((y_1, y_2, \frac{y_3}{g}), e_1)$ 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 Σ_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 Σ_L on inputs (y_1, y_2, y_3) and $(y_1, y_2, \frac{y_3}{u})$, respectively, and accepts or rejects accordingly.

For the case where $((y_1, y_2, \frac{y_3}{u}), (r, s)) \in R_L$, P computes (a_1, a_2, a_3) according to Σ_L, using $x = (y_1, y_2, \frac{y_3}{u})$ 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 Σ_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 Σ_{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 Σ_L with the same first message but with distinct challenges. Since Σ_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 Σ_{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 $((y_1, y_2, \frac{y_3}{u}), e_1)$ 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 $(y_1, y_2, y_3) \in L_{R_{OR}}$, and in particular that $(y_1, y_2, y_3) \in L_{R_L}$. In this case, it follows from perfect honest-verifier zero-knowledge of Σ_L that the conversation produced in step 2 is identically distributed as the corresponding conversation produced by P and V in Σ_{OR}. As for the conversation in step 3, the corresponding conversation in Σ_{OR} is also produced by S_L on input $((y_1, y_2, \frac{y_3}{u}), e_1)$, so it is clear that these conversations are identically distributed. A similar argument holds if $(y_1, y_2, \frac{y_3}{u}) \in L_{R_L}$. It follows that conversations output by S_{OR} are identically distributed as conversations between honest P and V in Σ_{OR}, and we conclude that Σ_{OR} has perfect honest-verifier zero-knowledge.

Appendix D. Proof of Theorem 7

Correctness of *compile*(Σ_{OR}) is straightforward to verify given that Σ_{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 Σ_{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 $r_i \in \mathbf{Z}_n^*$, computes $c_i \leftarrow Enc_P(z_i, r_i)$ and outputs $\pi = (a, (c_1, \ldots, c_6))$. It now follows from perfect honest-verifier zero-knowledge of Σ_{OR} that *compile*(Σ_{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, π_i), $1 \leq i \leq m(k)$, for some polynomial m. Run $V(1^\tau, x, \pi_i, sk_{KS})$, $1 \leq i \leq m(k)$. 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, π_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 τ. 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, $\mathit{compile}(\Sigma_{OR})$ is sound for $\mathcal{O}(\log\tau)$ 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 Σ_{OR}. Assume that there is a dishonest prover $\tilde{P}$ breaking the soundness of $\mathit{compile}(\Sigma_{OR})$. We start by outlining the strategy of the proof: We will use $\tilde{P}$ to obtain a fake-proof generator $\tilde{\mathcal{G}}_{\Sigma_{OR}} = \langle \tilde{G}_{\Sigma_{OR}}, \tilde{g}_{\Sigma_{OR}}\rangle$ for Σ_{OR}. Then, using $\tilde{P}$ and an algorithm A that completely breaks $\tilde{\mathcal{G}}$ on instances of size τ, we will construct an algorithm A' that breaks $\mathcal{H}_{\text{Paillier}}$ on instances of size 2τ, 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 Σ_{OR}.

The algorithm $\tilde{G}_{\Sigma_{OR}}$ takes 1^τ as input and generates a public key (pk, c) according to the protocol. $\tilde{G}_{\Sigma_{OR}}$ then runs $\tilde{P}$ on input $(1^\tau, (pk, c))$. Whenever $\tilde{P}$ outputs a collection of statement/proof pairs (x_i, π_i), where $1 \leq i \leq m(\tau)$, for some polynomial m, $\tilde{G}_{\Sigma_{OR}}$ replies with a random bit. When $\tilde{P}$ stops, $\tilde{G}_{\Sigma_{OR}}$ uniformly chooses one of the statement/proof pairs, say (x, π), which will be on the form $(x, (a, (c_1, \dots, c_6)))$, and outputs (x, a).

Note that, if $\tilde{P}$ outputs $\mathcal{O}(\log\tau)$ collections of statement/proof pairs, there is a non-negligible probability that all of the bits sent to $\tilde{P}$ by $\tilde{G}_{\Sigma_{OR}}$ 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, π_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 $\tilde{G}_{\Sigma_{OR}}$ chooses this particular pair to produce his output. Hence, with overall non-negligible probability, $\tilde{G}_{\Sigma_{OR}}$ 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 Σ_{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 $\tilde{g}_{\Sigma_{OR}}(x, a)$ be this unique e if such an e exists and $x \notin L_R$, and 0^τ otherwise, we see that $\tilde{\mathcal{G}}_{\Sigma_{OR}} = \langle \tilde{G}_{\Sigma_{OR}}, \tilde{g}_{\Sigma_{OR}}\rangle$ defines a fake-proof generator for Σ_{OR}. Accordingly, Assumption 6 implies in particular that $\mathcal{H}_{\text{Paillier}}$ is 2-harder than $\tilde{\mathcal{G}}_{\Sigma_{OR}}$.

Finally, given A that completely breaks $\tilde{\mathcal{G}}_{\Sigma_{OR}}$ on instances of size τ in time $T(\tau)$, we construct A' breaking $\mathcal{H}_{\text{Paillier}}$ on instances of size 2τ as follows: On input a 2τ-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 $\tilde{G}_{\Sigma_{OR}}$. 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 $T(\tau) + \mathit{poly}(\tau)$, we have a contradiction to the assumption that $\mathcal{H}_{\text{Paillier}}$ is 2-harder than any fake-proof generator for Σ_{OR}.

Appendix E. Proof of Theorem 8

Proof. We assume that there is an adversary A^{nf}_{BS} against the non-forgeability of our blind signature scheme, that engages in l protocols where *Signer* outputs *completed*, runs in time t and has advantage ϵ, and construct an adversary A^{nf}_{W} 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 $q' = l = \mathcal{O}(\log \tau)$ and ρ is a negligible function in τ.

A^{nf}_{W} 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^{nf}_{BS} on input $(1^\tau, crs, pk_{KS}, pk_W)$.

When A^{nf}_{BS} outputs a pair (c, π), A^{nf}_{W} verifies π 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 $\mathcal{O}^S$, and receives (σ_1^*, σ_2^*). He lets $\sigma_1' = \sigma_1^* \sigma_2^{*t}$, $\sigma_2' = \sigma_2^*$, and sends (σ_1', σ_2') to A^{nf}_{BS}. Note that, since $(\sigma_1^*, \sigma_2^*) = (g_2^\alpha (u' \prod_{i=1}^n u_i^{m_i})^r, g^r)$, for some randomly chosen $r \in \{0, \ldots, p-1\}$, we have $(\sigma_1', \sigma_2') = (g_2^\alpha (u' g^t \prod_{i=1}^n u_i^{m_i})^r, g^r)$. Moreover, since the verification procedure involves checking that $Enc_L(c, r^*, s^*) = \prod_{i=0}^{l-i} T_i^{2^i} \prod_{i=1}^n M_i$, it is guaranteed that $c = g^t \prod_{i=1}^n u_i^{m_i}$, so (σ_1', σ_2') has the right form.

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

We observe that A^{nf}_{BS}'s environment in $\mathbf{Exp}^{\mathrm{nf}}_{BS,A}(\tau)$ 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^{nf}_{BS} on input $(1^\tau, crs, pk_{KS}, pk_{BS})$.

When A^{nf}_{BS} outputs a pair (c, π), $\tilde{P}$ checks that $Enc_L(c, r^*, s^*) = \prod_{i=0}^{l-i} T_i^{2^i} \prod_{i=1}^n M_i$ exactly as *Signer* would do in the protocol, and outputs the statement/proof pairs (T_i, π_{T_i}), $0 \le i \le \tau - 1$, and $(M_i, \pi_{M_i}), 1 \le i \le n$. He accepts the proof only if he receives 1. He constructs (σ_1', σ_2') exactly as *Signer* would do in the protocol. We observe that, until the point where E occurs, A^{nf}_{BS}'s environment in $\mathbf{Exp}^{\mathrm{nf}}_{BS,A}(\tau)$ is perfectly simulated. Moreover, when E occurs, $\tilde{P}$ outputs some statement/proof pair, say (x^*, π^*), such that $x^* \notin L_{R_{OR}}$, 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

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

where ρ is a negligible function in τ.

Assume that A^{nf}_{BS} outputs message/signature pairs $(m_i, \sigma(m_i))$, $1 \le i \le k$, after A^{nf}_{W}, acting as *Signer*, has output *completed* l times, and assume that A^{nf}_{BS}

wins, that is, all the signatures are valid, no two messages are equal, and $k > l$. This means that A_W^{nf} has sent l queries to its signing oracle. Consequently, there is a pair, say (m^*, σ^*), where m^* has not been queried to the signing oracle. A_W^{nf} outputs (m^*, σ^*), and we conclude that, unless E occurs, and if A_{BS}^{nf} wins, then A_W^{nf} wins. Hence we get

$$\mathbf{Succ}_{BS,A}^{\mathrm{nf}} \geq (1-\rho)\epsilon,$$

and the proof is complete. □

Appendix F. Proof of Theorem 9

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 ϵ, and construct an adversary A_L^{ror} against the ROR-CPA security of the LE scheme, that queries its real-or-random oracle once, runs in time $t + \mathit{poly}(\tau)$, and has advantage at least $\frac{\epsilon}{2n+2\tau}$. To this end, we define a series of games as follows:

Game 0: A_{BS}^b runs in the experiment $\mathbf{Exp}_{BS,A}^{\mathrm{b}}(\tau)$, where $b = 0$.

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 crs, and runs A_{BS}^b on input $(1^\tau, \mathit{crs})$. When A_{BS}^b outputs $(\mathit{pk}, r, (m_0, m_1))$, P runs $\mathit{KS}(1^\tau, r)$ and obtains $(\mathit{sk}_{KS}, \mathit{pk}_{KS})$. He simulates $\mathit{User}(\mathit{pk}, m_0)$ and $\mathit{User}(\mathit{pk}, m_1)$, and if $\mathit{User}(\mathit{pk}, m_0)$ outputs $\sigma(m_0)$ and $\mathit{User}(\mathit{pk}, m_1)$ 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

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

Game 2: This game is the same as Game 1, with the following modification: When acting as $\mathit{User}(\mathit{pk}, m_0)$ and $\mathit{User}(\mathit{pk}, m_1)$, instead of honestly generating the proofs π_{T_i} and π_{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 $\mathit{compile}(\Sigma_{OR})$ has perfect zero-knowledge, proofs generated by M has exactly the same distribution as honestly generated proofs, so we have

$$\Big|\Pr[G_2 = 1] - \Pr[G_1 = 1]\Big| = 0.$$

Game 3: This game is the same as Game 2, with the following modification: When acting as $\mathit{User}(\mathit{pk}, m_0)$ and $\mathit{User}(\mathit{pk}, m_1)$, 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 $C = \mathit{Enc}_L(g^t \prod_{i=1}^n u_i^{m_i}, r^*, s^*)$. The commitments $T_o, \ldots, T_{\tau-1}$ and $M_1, \ldots, M_{n-1}$ are generated exactly as before, while

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

Observe that in Game 3 we have

$$
\begin{aligned}
M_n =&(\alpha_1^{r^*-\sum_{i=0}^{\tau-1} r_i 2^i - \sum_{i=1}^{n-1} r'_i}, \alpha_2^{s^*-\sum_{i=0}^{\tau-1} s_i 2^i - \sum_{i=1}^{n-1} s'_i}, \\
&\beta^{(r^*-\sum_{i=0}^{\tau-1} r_i 2^i - \sum_{i=1}^{n-1} r'_i)+(s^*-\sum_{i=0}^{\tau-1} s_i 2^i - \sum_{i=1}^{n-1} s'_i)} u_n^{m_n}).
\end{aligned}
$$

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

$$\Big|\Pr[G_3 = 1] - \Pr[G_2 = 1]\Big| = 0.$$

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 $\rho_0 \in \mathbf{G}$ and lets $T_0 = Enc_L(\rho_0, r_0, s_0)$.

In each of the games Game 5, ..., Game $\tau + 3$, one more of the g^{t_i} is replaced by a random element in $\mathbf{G}$. Correspondingly, in each of the games Game $\tau + 4$, ..., Game $n+\tau+2$, 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, π_1) and (c_2, π_2) denote the messages sent by *User* in the first and the second protocol, respectively. The *view* of A_{BS}^b 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}^b's view in Game $n + l + 3$ is negligibly close to the distribution of A_{BS}^b's view in Game $n + l + 2$. First, we note that the proofs π_1 and π_2 are now independent of the message input to *User*, i.e. (π_1, π_2) has exactly the same distribution in both games. Also, in both games, c_1 and c_2 are both on the form $g^t \prod_{i=1}^n u_i^{m_{b,i}}$ for some bit 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 (π_1, π_2), so the distribution of $((c_1, c_2), (\pi_1, \pi_2))$ is exactly the same in both games.

Assume that A_{BS}^b's response (σ'_1, σ'_2) has the correct form in both protocols, i.e. in the first protocol $\sigma'_1 = g_2^\alpha (u' c_1)^r$ 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 $(g_2^\alpha (u' \prod_{i=1}^n u_i^{m_{0,i}})^{r_1 + r'_1}, g^{r_1 + r'_1})$ and $(g_2^\alpha (u' \prod_{i=1}^n u_i^{m_{1,i}})^{r_2 + r'_2}, g^{r_2 + r'_2})$, 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 (π_1, π_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^b_{BS}'s response (σ'_1, σ'_2) does not have the correct form. For instance, assume that in the first protocol $\sigma'_1 = g_2^\alpha(u'c_1)^r$ 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 $u'c \neq 1$. Thus, A^b_{BS}'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

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

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 $\rho_0 \in \mathbf{G}$ and letting $T_0 = Enc_L(\rho_0, r_0, s_0)$, P lets $T_0 = Enc_L(g^{t_0}, r_0, s_0)$.

In each of the games Game $n + \tau + 5, \ldots,$ Game $n + 2\tau + 3$, one more of the T_i is generated by encrypting g^{t_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 $u_i^{m_i}$ 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

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

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 π_{T_i} and π_{M_i}, as described in Game 2, P generates these proofs according to the protocol.

Again, since *compile*(Σ_{OR}) has perfect zero-knowledge, we have

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

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

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

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

We now construct an adversary A^{ror}_L trying to break the real-or-random blindness of linear encryption. A^{ror}_L 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^{ror} then behaves exactly as P in Game 3, with the following exceptions: A_L^{ror} lets $crs = \alpha_1 || \alpha_2 || \beta$, and instead of generating T_0 as in Game 3, A_L^{ror} sends g^{t_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^{ror} outputs b'.

We observe that, if A_L^{ror}'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

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

Defining the behavior of A_L^{ror} 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

$$\mathbf{Adv}^{\mathrm{ror}}_{\mathrm{LE},A^{\mathrm{ror}}}(\tau) \geq \frac{1}{2n+2\tau}\left(\sum_{i=3}^{n+\tau+1}\Big|\Pr[G_i = 1] - \Pr[G_{i+1} = 1]\Big| + \sum_{i=n+\tau+3}^{2n+2\tau+1}\Big|\Pr[G_{i+1} = 1] - \Pr[G_i = 1]\Big|\right).$$

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

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

Hence we may write

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{ror}}_{\mathrm{LE},A^{\mathrm{ror}}}(\tau) &\geq \frac{1}{2n+2\tau}\sum_{i=0}^{2n+2\tau+5}\Big|\Pr[G_{i+1} = 1] - \Pr[G_i = 1]\Big| \\
&\geq \frac{1}{2n+2\tau}\Big|\sum_{i=0}^{2n+2\tau+5}\Pr[G_{i+1} = 1] - \Pr[G_i = 1]\Big| \\
&= \frac{1}{2n+2\tau}\Big|\Pr[G_{2n+2\tau+5} = 1] - \Pr[G_0 = 1]\Big| \\
&= \frac{1}{2n+2\tau}\mathbf{Adv}^{\mathrm{b}}_{\mathrm{BS},A^b}(\tau),
\end{aligned}$$

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, $\mathcal{F}_{\text{AOS}}$, 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 an anonymous secure channel. This functionality, which is denoted by $\mathcal{F}_{\text{ASC}}$, 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 $\mathcal{F}_{\text{ASC}}$ 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 $\mathcal{F}_{\text{ASC}}$, it may happen that the ideal adversary receives two messages on the form $(\textbf{ASC.Send}, \textit{sid}, \textit{ssid}, P_i, \textit{partner}, |m|)$ before a message $(\textbf{ASC.Send}, \textit{sid}, \textit{ssid}, P_i, \textit{partner}, \textit{ack})$ is sent in the opposite direction. In this case, upon the first $(\textbf{ASC.Send}, \textit{sid}, \textit{ssid}, P_i, \textit{partner}, \textit{ack})$ 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 renames 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 $\mathcal{F}_{\text{CA}}$ used by Nagao et al. is replaced by an anonymized variant, $\mathcal{F}_{\text{ACA}}$, 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 π_{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 $\mathcal{F}_{\text{ASC}}$ ensures that messages sent across the channel arrive at the recipient in the correct order, the protocol π_{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. *π_{ASC} securely realizes $\mathcal{F}_{ASC}$ in the ($\mathcal{F}_{ACA}$, $\mathcal{F}_{KEM\text{-}DEM}$, $\mathcal{F}_{AN}$)-hybrid model under static corruption.*

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

As usual, $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and forwards all messages from $\mathcal{Z}$ to $\mathcal{A}$ and back. When $\mathcal{A}$ corrupts a party P, $\mathcal{S}$ corrupts $\tilde{P}$. When $\tilde{P}$ is corrupt, any input from $\mathcal{Z}$ meant for $\tilde{P}$ goes directly to $\mathcal{S}$, who forwards it to $\mathcal{A}$ on the input tape corresponding to P, and the other way around. Moreover, $\mathcal{S}$ can send messages to $\mathcal{F}$ in the name of $\tilde{P}$, and messages from $\mathcal{F}$ meant for $\tilde{P}$ go to $\mathcal{S}$. 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: $\mathcal{S}$ simulates honest P_i, honest P_j and functionalities $\mathcal{F}_{\text{ACA}}$, $\mathcal{F}_{\text{KEM-DEM}}$ and $\mathcal{F}_{\text{AN}}$.

Simulating the set-up: Upon receiving $(\mathbf{ASC}.\mathbf{Register}, \mathit{sid}_1, P_i)$ from $\mathcal{F}_{\mathrm{ASC}}$, $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$ sends $(\mathbf{KEM}.\mathbf{KeyGen}, \mathit{sid}_2)$ to $\mathcal{A}$, and waits to receive $(\mathbf{KEM}.\mathbf{Key}, \mathit{sid}_2, \mathit{pk}_i)$ from $\mathcal{A}$. Then $\mathcal{S}|\mathcal{F}_{\mathrm{ACA}}$ sends $(\mathbf{ACA}.\mathbf{Register}, \mathit{sid}_3, P_i, \mathit{pk}_i)$ to $\mathcal{A}$. Upon receiving $(\mathbf{ACA}.\mathbf{Register}, \mathit{sid}_3, \mathit{ack})$ from $\mathcal{A}$, $\mathcal{S}|\mathcal{F}_{\mathrm{ACA}}$ stores (P_i, pk_i) and sends $(\mathbf{ASC}.\mathbf{Registered}, \mathit{sid}_1, P_i, \mathit{ack})$ to $\mathcal{F}_{\mathrm{ASC}}$.

Simulating the session set-up: Upon receiving $(\mathbf{ASC}.\mathbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i)$ from $\mathcal{F}_{\mathrm{ASC}}$, $\mathcal{S}$ chooses a random identity ID and stores $(\mathit{ID}, \mathit{ssid})$. Then $\mathcal{S}|\mathcal{F}_{\mathrm{ACA}}$ sends $(\mathbf{ACA}.\mathbf{Retrieve}, \mathit{sid}_3, P_i)$ to $\mathcal{A}$. Upon receiving $(\mathbf{ACA}.\mathbf{Retrieve}, \mathit{sid}_3, \mathit{ack})$ from $\mathcal{A}$, $\mathcal{S}|\mathcal{F}_{\mathrm{ACA}}$ checks that there is a stored entry (P_i, pk_i). $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$ then sends $(\mathbf{KEM}.\mathbf{Encrypt}, \mathit{sid}_2, \mathit{ssid}, \mathit{pk}_i)$ to $\mathcal{A}$, waits to receive $(\mathbf{EncryptedSharedKey}, \mathit{sid}_2, \mathit{ssid}, \mathit{pk}_i, C_0)$ from $\mathcal{A}$, and stores $(\mathit{ID}, \mathit{ssid}, C_0, \mathit{active})$. $\mathcal{S}|\mathcal{F}_{\mathrm{AN}}$ then sends $(\mathbf{AN}.\mathbf{Send}, (\mathit{sid}_4, \mathit{ssid}), (\mathbf{ASC}.\mathbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, C_0, \mathit{ID}), \mathit{ID}, P_i)$ to $\mathcal{A}$, and $\mathcal{S}$ sends $(\mathbf{ASC}.\mathbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\mathrm{ASC}}$.
If $\mathcal{S}|\mathcal{F}_{\mathrm{AN}}$ receives $(\mathbf{AN}.\mathbf{Send}, (\mathit{sid}_4, \mathit{ssid}), m', \mathit{ID}')$ from $\mathcal{A}$, where ID' represents a corrupt party, then $\mathcal{S}|\mathcal{F}_{\mathrm{AN}}$ sends $(\mathbf{AN}.\mathbf{Receive}, (\mathit{sid}_4, \mathit{ssid}), m')$ to $\mathcal{A}$ on the input tape corresponding to ID'. Otherwise, if $\mathit{ID}' = P_i$, and m' is on the form $(\mathbf{ASC}.\mathbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, C_0^*, \mathit{ID}^*)$, then $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$ sends $(\mathbf{KEM}.\mathbf{Decrypt}, \mathit{sid}_2, \mathit{ssid}, C_0^*)$ to $\mathcal{A}$. Upon receiving $(\mathbf{KEM}.\mathbf{Decrypt}, \mathit{sid}_2, \mathit{ssid}, \mathit{ack})$ from $\mathcal{A}$, $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$ stores $(P_i, \mathit{ssid}, C_0^*, \mathit{active})$, and $\mathcal{S}$ sends $(\mathbf{ASC}.\mathbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i, \mathit{ack})$ to $\mathcal{F}_{\mathrm{ASC}}$.

Simulating the data exchange: Upon receiving $(\mathbf{ASC}.\mathbf{Send}, \mathit{sid}_1, \mathit{ssid}, P_i, \mathit{partner}, |M|)$ from $\mathcal{F}_{\mathrm{ASC}}$, $\mathcal{S}$ checks that there is a stored entry $(P_i, \mathit{ssid}, C_0^*, \mathit{active})$ 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. $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$ then sends $(\mathbf{DEM}.\mathbf{Encrypt}, \mathit{sid}_2, \mathit{ssid}, |m|)$ to $\mathcal{A}$, and waits to receive $(\mathbf{DEM}.\mathbf{Ciphertext}, \mathit{sid}_2, \mathit{ssid}, c)$ from $\mathcal{A}$. $\mathcal{S}|\mathcal{F}_{\mathrm{AN}}$ then sends $(\mathbf{AN}.\mathbf{Send}, (\mathit{sid}_4, \mathit{ssid}), c, P_i, \mathit{ID})$ to $\mathcal{F}_{\mathrm{AN}}$. The next time $\mathcal{S}|\mathcal{F}_{\mathrm{AN}}$ receives a message $(\mathbf{AN}.\mathbf{Send}, (\mathit{sid}_4, \mathit{ssid}), c', \mathit{ID}')$ from $\mathcal{A}$, if ID' represents a corrupt party, then $\mathcal{S}|\mathcal{F}_{\mathrm{AN}}$ sends $(\mathbf{AN}.\mathbf{Receive}, (\mathit{sid}_4, \mathit{ssid}), c')$ to $\mathcal{A}$ on the input tape corresponding to ID'. Otherwise, if $\mathit{ID}' = \mathit{ID}$, and there is a stored entry $(\mathit{ID}, \mathit{ssid}, C_0, \mathit{active})$ for some C_0, then $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$ sends $(\mathbf{DEM}.\mathbf{Decrypt}, \mathit{sid}_2, \mathit{ssid}, c')$ to $\mathcal{A}$, and waits to receive $(\mathbf{DEM}.\mathbf{Plaintext}, \mathit{sid}_2, \mathit{ssid}, m')$ from $\mathcal{A}$. If $C_0^* = C_0$ and $c' = c$, then $\mathcal{S}$ sends $(\mathbf{ASC}.\mathbf{Send}, \mathit{sid}_1, \mathit{ssid}, P_i, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\mathrm{ASC}}$.

Upon receiving $(\mathbf{ASC.Send}, sid_1, ssid, partner, P_i, |M|)$ from $\mathcal{F}_{\text{ASC}}$, $\mathcal{S}$ 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. $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$ then sends $(\mathbf{DEM.Encrypt}, sid_2, ssid, |m|)$ to $\mathcal{A}$, and waits to receive $(\mathbf{DEM.Ciphertext}, sid_2, ssid, c)$ from $\mathcal{A}$. $\mathcal{S}|\mathcal{F}_{\text{AN}}$ then sends $(\mathbf{AN.Send}, (sid_4, ssid), c, ID, P_i)$ to $\mathcal{F}_{\text{AN}}$. The next time $\mathcal{S}|\mathcal{F}_{\text{AN}}$ receives a message $(\mathbf{AN.Send}, (sid_4, ssid), c', ID')$ from $\mathcal{A}$, if ID' represents a corrupt party, then $\mathcal{S}|\mathcal{F}_{\text{AN}}$ sends $(\mathbf{AN.Receive}, (sid_4, ssid), c')$ to $\mathcal{A}$ 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 $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$ sends $(\mathbf{DEM.Decrypt}, sid_2, ssid, c')$ to $\mathcal{A}$, and waits to receive $(\mathbf{DEM.Plaintext}, sid_2, ssid, m')$ from $\mathcal{A}$. If $C_0 = C_0^*$ and $c' = c$, then $\mathcal{S}$ sends $(\mathbf{ASC.Send}, sid_1, ssid, partner, P_i, ack)$ to $\mathcal{F}_{\text{ASC}}$.

Simulating the session ending: Upon receiving $(\mathbf{ASC.ExpireSession}, sid_1, ssid, P_i/partner)$ from $\mathcal{F}_{\text{ASC}}$, $\mathcal{S}$ 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 $\mathcal{S}|\tilde{P}_i$ are forwarded to $A|P_i$, and outputs from $A|P_i$ are output by $\mathcal{S}|\tilde{P}_i$. $\mathcal{S}$ simulates honest P_j and functionalities $\mathcal{F}_{\text{ACA}}$, $\mathcal{F}_{\text{KEM-DEM}}$ and $\mathcal{F}_{\text{AN}}$. $\mathcal{S}$ can perfectly simulate $\mathcal{F}_{\text{ACA}}$, since all the information obtained by $\mathcal{F}_{\text{ACA}}$ in the real protocol is obtained by $\mathcal{S}$ in the simulated protocol. Regarding $\mathcal{F}_{\text{AN}}$, $\mathcal{S}$ does not obtain P_j's identity, but this is also perfectly hidden from $\mathcal{A}$, so $\mathcal{S}$ can still perfectly simulate $\mathcal{A}$'s view in the real protocol. As for $\mathcal{F}_{\text{KEM-DEM}}$, when P_i is corrupt, $\mathcal{S}$ 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, $\mathcal{S}$ can perfectly simulate the messages expected by A from $\mathcal{F}_{\text{KEM-DEM}}$. In the following, when we say that $A|P_i$ or $\mathcal{S}|ID^*$ sends a message to $\mathcal{S}|\mathcal{F}$, where $\mathcal{F}$ is one of the above functionalities, this implies that $\mathcal{S}$ proceeds according to the description of this functionality, and allows $\mathcal{A}$ to delay messages, determine keys, redirect messages etc. whenever appropriate.

Simulating the set-up: The first time $\mathcal{S}|\tilde{P}_i$ receives some input, $\mathcal{S}|\tilde{P}_i$ sends $(\mathbf{ASC.Register}, sid_1)$ to $\mathcal{F}_{\text{ASC}}$, and upon receiving $(\mathbf{ASC.Register}, sid_1, P_i)$ from $\mathcal{F}_{\text{ASC}}$, $\mathcal{S}$ sends $(\mathbf{ASC.Register}, sid_1, ack)$ to $\mathcal{F}_{\text{ASC}}$.

Simulating the session set-up: Upon receiving $(\mathbf{ASC.EstablishSession}, sid_1, ssid, P_i)$ from $\mathcal{F}_{\text{ASC}}$, $\mathcal{S}$ sets a variable $P_i - ssid - sn$ equal to 1, and sends $(\mathbf{ASC.EstablishSession}, sid_1, ssid, P_i, ack)$ to $\mathcal{F}_{\text{ASC}}$. $\mathcal{S}$ chooses a random identity ID^*, stores $(ID^*, ssid)$, and $\mathcal{S}|ID^*$ sends $(\mathbf{AN.Register}, (sid_4, ssid))$ to $\mathcal{S}|\mathcal{F}_{\text{AN}}$. Upon receiving $(\mathbf{AN.Registered}, (sid_4, ssid), ID)$ from $\mathcal{S}|\mathcal{F}_{\text{AN}}$, $\mathcal{S}|ID^*$ sends $(\mathbf{ACA.Retrieve}, sid_3, P_i)$ to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$.

Upon receiving $(\mathbf{ACA.Retrieved}, \mathit{sid}_3, P_i, \mathit{pk}_i)$ from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}|\mathit{ID}^*$ sends $(\mathbf{KEM.Encrypt}, \mathit{sid}_2, \mathit{ssid}, \mathit{pk}_i)$ to $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$. Upon receiving $(\mathbf{EncryptedSharedKey}, \mathit{sid}_2, \mathit{ssid}, \mathit{pk}_i, C_0)$ from $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$, $\mathcal{S}|\mathit{ID}^*$ stores $(\mathit{ID}^*, \mathit{ssid}, \mathit{active})$, and sets a variable $\mathit{ID}^* - \mathit{ssid} - \mathit{sn}$ equal to 1. $\mathcal{S}|\mathit{ID}^*$ then sends $(\mathbf{AN.Send}, (\mathit{sid}_4, \mathit{ssid}), (\mathbf{ASC.EstablishSession}, \mathit{sid}_1, C_0, \mathit{ID}), P_i)$ to $\mathcal{S}|\mathcal{F}_{\text{AN}}$, and sends $(\mathbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\text{ASC}}$.

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

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

When P_i is honest, while P_j is corrupt: As explained earlier, inputs to $\mathcal{S}|\tilde{P}_j$ are forwarded to $A|P_j$, and outputs from $A|P_j$ are output by $\mathcal{S}|\tilde{P}_j$. $\mathcal{S}$ simulates honest P_i and functionalities $\mathcal{F}_{\text{ACA}}$, $\mathcal{F}_{\text{KEM-DEM}}$ and $\mathcal{F}_{\text{AN}}$. $\mathcal{S}$ 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 $\mathcal{S}|P_i$ sends a message to $\mathcal{S}|\mathcal{F}$, where $\mathcal{F}$ is one of the above functionalities, this implies that $\mathcal{S}$ proceeds according to the description of this functionality.

Simulating the set-up: Upon receiving $(\mathbf{ASC.Register}, \mathit{sid}_1)$ from $\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|P_i$ sends $(\mathbf{KEM.KeyGen}, \mathit{sid}_2)$ to $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$. Upon receiving $(\mathbf{KEM.Key}, \mathit{sid}_2, \mathit{pk}_i)$ from $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$, $\mathcal{S}|P_i$ sends $(\mathbf{ACA.Register}, \mathit{sid}_3, \mathit{pk}_i)$ to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$. Upon receiving $(\mathbf{ACA.Registered}, \mathit{sid}_3)$ from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}$ sends $(\mathbf{ASC.Register}, \mathit{sid}_1, \mathit{ack})$ to $\mathcal{F}_{\text{ASC}}$.

Simulating the session set-up: When $A|P_j$ successfully establishes a session with P_i (i.e. when $\mathcal{S}|P_i$ receives $(\mathbf{AN.Receive}, (\mathit{sid}_4, \mathit{ssid}), (\mathbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, C_0', \mathit{ID}'))$ from $\mathcal{F}_{\mathrm{AN}}$, $\mathcal{S}|P_i$ sends $(\mathbf{KEM.Decrypt}, \mathit{sid}_2, \mathit{ssid}, C_0')$ to $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}}$, and $(\mathbf{KEM.Decrypt}, \mathit{sid}_2, \mathit{ssid}, \mathit{ok})$ is returned from $\mathcal{S}|\mathcal{F}_{\text{KEM-DEM}})$, $\mathcal{S}$ stores $(P_i, \mathit{ssid}, \mathit{active})$ and sets variables $P_i - \mathit{ssid} - \mathit{sn}$ and $\mathit{ID}' - \mathit{ssid} - \mathit{sn}$ equal to 1. $\mathcal{S}|\tilde{P}_j$ then sends $(\mathbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i)$ to $\mathcal{F}_{\mathrm{ASC}}$. Upon receiving $(\mathbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i)$ from $\mathcal{F}_{\mathrm{ASC}}$, $\mathcal{S}$ sends $(\mathbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\mathrm{ASC}}$, and then $\mathcal{S}$ sends $(\mathbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i, \mathit{ack})$ to $\mathcal{F}_{\mathrm{ASC}}$.

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

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

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

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

We conclude that π_{ASC} securely realizes $\mathcal{F}_{\mathrm{ASC}}$ in the ($\mathcal{F}_{\text{KEM-DEM}}$, $\mathcal{F}_{\mathrm{ACA}}$, $\mathcal{F}_{\mathrm{AN}}$)-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 $\mathcal{F}_{\mathrm{AOS}}$. In addition to the functionalities $\mathcal{F}_{\mathrm{ASC}}$ and $\mathcal{F}_{\mathrm{ACA}}$ 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, $\mathcal{F}_{\mathrm{ABS}}$, 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 $\mathcal{F}^*_{\mathrm{ASC}}$ denote a functionality for a secure channel incorporating these properties. Intuitively, $\mathcal{F}^*_{\mathrm{ASC}}$ can be realized using $\mathcal{F}_{\mathrm{ASC}}$ in combination with $\mathcal{F}_{\mathrm{ACA}}$ 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 $\mathcal{F}^*_{\mathrm{ASC}}$, and then have the user identify himself by signing a randomly chosen challenge.

Obviously, in order to realize $\mathcal{F}_{\mathrm{ABS}}$, 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 $\mathcal{F}_{\mathrm{ABS}}$ must be modified in order to obtain a realizing protocol. Anyway, the resulting functionality can still be used to realize $\mathcal{F}_{\mathrm{AOS}}$.

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 $\mathcal{F}_{\mathrm{ABS}}$, $\mathcal{F}_{\mathrm{ACA}}$, $\mathcal{F}_{\mathrm{SC}}$ and $\mathcal{F}_{\mathrm{ASC}}$ constitute the building blocks necessary to construct our hybrid protocol π_{AOS} for an anonymous online service. A detailed description of the protocol is given in Figures 9 and 10.

Concerning the description of π_{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 π^*_{AOS} is the protocol π_{AOS} where $\mathcal{F}_{\mathrm{ABS}}$ 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 σ such that (c, σ) 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 σ on some challenge c', but does not send σ to Q. Instead, he starts a new session, and receives a new challenge c from Q, such that (c, σ) is accepted by the verification algorithm. Assume that B refuses to sign this time, and that P_i sends σ to Q. Since P_i's free signature count is positive at this point, the pair (c, σ) 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, π^*_{AOS} does not realize $\mathcal{F}_{\mathrm{AOS}}$.

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

Theorem 2. *π_{AOS} realizes $\mathcal{F}_{AOS}$ in the ($\mathcal{F}_{ABS}$, $\mathcal{F}_{ACA}$, $\mathcal{F}_{SC}$, $\mathcal{F}_{ASC}$)-hybrid model under static corruption.*

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

As usual, $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and forwards all messages from $\mathcal{Z}$ to $\mathcal{A}$ and back. When $\mathcal{A}$ corrupts a party P, $\mathcal{S}$ corrupts $\tilde{P}$. When $\tilde{P}$ is corrupt, any input from $\mathcal{Z}$ meant for $\tilde{P}$ goes directly to $\mathcal{S}$, who forwards it to $\mathcal{A}$ on the input tape corresponding to P, and the other way around. Moreover, $\mathcal{S}$ can send messages to $\mathcal{F}$ in the name of $\tilde{P}$, and messages from $\mathcal{F}$ meant for $\tilde{P}$ go to $\mathcal{S}$. 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, Q and B are honest: $\mathcal{S}$ simulates honest P_i, 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 $\mathcal{S}$ obtains all the information needed to perfectly simulate the above functionalities. Note that, it does not matter that $\mathcal{S}$ does not obtain P_i's identity or the content of messages sent between Q and P_i, since this information is also perfectly hidden from $\mathcal{A}$ in the real protocol. In the following, when we say that $\mathcal{S}|P$ sends a message to $\mathcal{S}|\mathcal{F}$, where $\mathcal{F}$ is one of the above functionalities and P is either P_i, Q or B, this implies that $\mathcal{S}$ proceeds according to the description of this functionality, and allows $\mathcal{A}$ to delay messages, determine keys etc. whenever appropriate.

Simulating the set-up: Upon receiving ($\mathbf{AOS}.\mathbf{Register}, \mathit{sid}_1$) from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}|B$ sends ($\mathbf{ABS}.\mathbf{KeyGen}, \mathit{sid}_2$) to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. Upon receiving ($\mathbf{ABS}.\mathbf{Key}, \mathit{sid}_2, \mathit{pk}$) from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}|B$ sends ($\mathbf{ACA}.\mathbf{Register}, \mathit{sid}_3, \mathit{pk}$) to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$. Upon receiving ($\mathbf{ACA}.\mathbf{Registered}, \mathit{sid}_3$) from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}|B$ sends ($\mathbf{SC}.\mathbf{Register}$, sid_4) to $\mathcal{S}|\mathcal{F}_{\text{SC}}$. Upon receiving ($\mathbf{SC}.\mathbf{Registered}, \mathit{sid}_4$) from $\mathcal{S}|\mathcal{F}_{\text{SC}}$, $\mathcal{S}$ sends ($\mathbf{AOS}.\mathbf{Register}, \mathit{sid}_1, \mathit{ack}$) to $\mathcal{F}_{\text{AOS}}$.
Upon receiving ($\mathbf{AOS}.\mathbf{Register}, \mathit{sid}_1, Q$) from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}|Q$ sends ($\mathbf{ACA}.\mathbf{Retrieve}, \mathit{sid}_3, B$) to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$. Upon receiving ($\mathbf{ACA}.\mathbf{Retrieved}$, $\mathit{sid}_3, B, \mathit{pk}$) from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}|Q$ sends ($\mathbf{ASC}.\mathbf{Register}, \mathit{sid}_5$) to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving ($\mathbf{ASC}.\mathbf{Registered}, \mathit{sid}_5$) from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}$ sends ($\mathbf{AOS}.\mathbf{Register}$, $\mathit{sid}_1, Q, \mathit{ack}$) to $\mathcal{F}_{\text{AOS}}$.

Simulating the session set-up: Upon receiving ($\mathbf{AOS}.\mathbf{EstablishSession}$, $\mathit{sid}_1, \mathit{ssid}, Q$) from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ chooses a random identity ID and stores ($\mathit{ID}, \mathit{ssid}$). $\mathcal{S}|\mathit{ID}$ sends ($\mathbf{ASC}.\mathbf{EstablishSession}, \mathit{sid}_5, \mathit{ssid}, Q$) to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving ($\mathbf{ASC}.\mathbf{SessionEstablished}, \mathit{sid}_5, \mathit{ssid}, Q$) from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|\mathit{ID}$ sends ($\mathbf{ACA}.\mathbf{Retrieve}, \mathit{sid}_3, B$) to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$. Upon receiving ($\mathbf{ACA}.\mathbf{Retrieved}, \mathit{sid}_3, B, \mathit{pk}$) from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}|\mathit{ID}$ sends ($\mathbf{ASC}.\mathbf{Send}$, $\mathit{sid}_5, \mathit{ssid}, \mathbf{AOS}.\mathbf{Pay}$) to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving ($\mathbf{ASC}.\mathbf{Receive}$, $\mathit{sid}_5, \mathit{ssid}, \mathbf{AOS}.\mathbf{Pay}$) from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}$ picks a random challenge c, and $\mathcal{S}|Q$ sends ($\mathbf{ASC}.\mathbf{Send}, \mathit{sid}_5, \mathit{ssid}, (\mathbf{AOS}.\mathbf{Challenge}, c)$) to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving ($\mathbf{ASC}.\mathbf{Receive}, \mathit{sid}_5, \mathit{ssid}, (\mathbf{AOS}.\mathbf{Challenge}, c)$) from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|\mathit{ID}$ sends ($\mathbf{ABS}.\mathbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{pk}, c$) to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. When $\mathcal{S}|B$ receives ($\mathbf{ABS}.\mathbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{ID}, \mathit{request}$) from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}$ sends ($\mathbf{AOS}.\mathbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, \mathit{ack}$) to $\mathcal{F}_{\text{AOS}}$.
If $\mathcal{S}$ receives ($\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no}$) from $\mathcal{F}_{\text{AOS}}$, then $\mathcal{S}|B$ sends ($\mathbf{ABS}.\mathbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{no}$) to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. Then, if $\mathcal{S}|\mathit{ID}$ receives ($\mathbf{ABS}.\mathbf{Sign}$, $\mathit{sid}_2, \mathit{ssid}, \mathit{no}$) from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}$ sends ($\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no}, \mathit{ack}$) to $\mathcal{F}_{\text{AOS}}$. Otherwise, if $\mathcal{S}$ receives ($\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{yes}$) from $\mathcal{F}_{\text{AOS}}$, then $\mathcal{S}|B$ sends ($\mathbf{ABS}.\mathbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{yes}$) to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. Then, if $\mathcal{S}|\mathit{ID}$ receives ($\mathbf{ABS}.\mathbf{Signature}, \mathit{sid}_2, \mathit{ssid}, \sigma$) from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, then $\mathcal{S}|\mathit{ID}$ sends ($\mathbf{ASC}.\mathbf{Send}$, $\mathit{sid}_5, \mathit{ssid}, (\mathbf{AOS}.\mathbf{Response}, \sigma)$) to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. $\mathcal{S}$ stores ($\mathit{ID}, \mathit{ssid}, \mathit{active}$) and sends ($\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}_2, \mathit{ssid}, \mathit{partner}, \mathit{ack}$) to $\mathcal{F}_{\text{AOS}}$.

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

Simulating the data exchange: Upon receiving $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner}, Q, |m|)$ from $\mathcal{F}_{\text{AOS}}$, if $(\mathit{ID}, \mathit{ssid}, \mathit{active})$ is stored, then $\mathcal{S}$ picks a random message m' of length $|m|$, and $\mathcal{S}|\mathit{ID}$ sends $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, m')$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. When $\mathcal{S}|Q$ receives $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, m')$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, if $(Q, \mathit{ssid}, \mathit{active})$ is stored, then $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner}, Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.
Upon receiving $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{partner}, |m|)$ from $\mathcal{F}_{\text{AOS}}$, if $(Q, \mathit{ssid}, \mathit{active})$ is stored, then $\mathcal{S}$ picks a random message m' of length $|m|$, and $\mathcal{S}|Q$ sends $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, m')$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. When $\mathcal{S}|\mathit{ID}$ receives $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, m')$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, if $(\mathit{ID}, \mathit{ssid}, \mathit{active})$ is stored, then $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.

Simulating the session ending: Upon receiving $(\textbf{AOS}.\textbf{ExpireSession}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner})$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ removes $(\mathit{ID}, \mathit{ssid}, \mathit{active})$, and $\mathcal{S}|\mathit{ID}$ sends $(\textbf{ASC}.\textbf{ExpireSession}, \mathit{sid}_5, \mathit{ssid})$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$.
Upon receiving $(\textbf{AOS}.\textbf{ExpireSession}, \mathit{sid}_1, \mathit{ssid}, Q)$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ removes $(Q, \mathit{ssid}, \mathit{active})$, and $\mathcal{S}|Q$ sends $(\textbf{ASC}.\textbf{ExpireSession}, \mathit{sid}_5, \mathit{ssid})$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$.

Simulating the collecting: Upon receiving $(\textbf{AOS}.\textbf{Collect}, \mathit{sid}_1, \mathit{ssid}', Q, n)$ from $\mathcal{F}_{\text{AOS}}$, if $\mathcal{S}$ has at least n stored pairs (c, σ), then $\mathcal{S}|Q$ sends $(\textbf{SC}.\textbf{EstablishSession}, \mathit{sid}_4, \mathit{ssid}', B)$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$. When $\mathcal{S}|Q$ receives $(\textbf{SC}.\textbf{SessionEstablished}, \mathit{sid}_4, \mathit{ssid}', B)$ from $\mathcal{S}|\mathcal{F}_{\text{SC}}$, $\mathcal{S}$ lets s be a set of n stored pairs (c, σ), and sends $(\textbf{SC}.\textbf{Send}, \mathit{sid}_4, \mathit{ssid}', (\textbf{AOS}.\textbf{Collect}, s))$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$. When $\mathcal{S}|B$ receives $(\textbf{SC}.\textbf{Receive}, \mathit{sid}_4, \mathit{ssid}', (\textbf{AOS}.\textbf{Collect}, s))$ from $\mathcal{S}|\mathcal{F}_{\text{SC}}$, for every pair (c, σ) in s, $\mathcal{S}$ checks that there is no stored pair (c, σ') for any σ', and $\mathcal{S}|B$ sends $(\textbf{ABS}.\textbf{Verify}, \mathit{sid}_2, \mathit{pk}, c, \sigma)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. If, for every pair, $\mathcal{S}|B$ receives $(\textbf{ABS}.\textbf{Verify}, \mathit{sid}_2, \mathit{accept})$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, then $\mathcal{S}$ stores s and $\mathcal{S}|B$ sends $(\textbf{SC}.\textbf{Send}, \mathit{sid}_4, \mathit{ssid}', \textbf{AOS}.\textbf{Collected})$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$. $\mathcal{S}|B$ then sends $(\textbf{SC}.\textbf{ExpireSession}, \mathit{sid}_4, \mathit{ssid}')$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$, and $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Collect}, \mathit{sid}_1, \mathit{ssid}', \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.
When $\mathcal{S}|Q$ receives $(\textbf{SC}.\textbf{Receive}, \mathit{sid}_4, \mathit{ssid}', \textbf{AOS}.\textbf{Collected})$ from $\mathcal{S}|\mathcal{F}_{\text{SC}}$, $\mathcal{S}$ removes the pairs (c, σ) contained in s. $\mathcal{S}|Q$ then sends $(\textbf{SC}.\textbf{ExpireSession}, \mathit{sid}_4, \mathit{ssid}')$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$, and $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Collect}, \mathit{sid}_1, \mathit{ssid}', Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.

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

When P_i is corrupt, while Q and B are honest: $\mathcal{S}$ simulates honest Q and B and functionalities $\mathcal{F}_{\mathrm{ABS}}$, $\mathcal{F}_{\mathrm{ACA}}$, $\mathcal{F}_{\mathrm{SC}}$ and $\mathcal{F}_{\mathrm{ASC}}$. We observe that $\mathcal{S}$ obtains all the information needed to perfectly simulate the above functionalities. In the following, when we say that an entity sends a message to $\mathcal{S}|\mathcal{F}$, where $\mathcal{F}$ is one of the above functionalities, this implies that $\mathcal{S}$ proceeds according to the description of this functionality, and allows $\mathcal{A}$ 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 $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, \textbf{AOS}.\textbf{Pay})$ from $\mathcal{S}|\mathcal{F}_{\mathrm{ASC}}$, $\mathcal{S}$ picks a random challenge c, and $\mathcal{S}|Q$ sends $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS}.\textbf{Challenge}, c))$ to $\mathcal{S}|\mathcal{F}_{\mathrm{ASC}}$.
When $\mathcal{S}|B$ receives $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, P_i, \mathit{request})$ from $\mathcal{S}|\mathcal{F}_{\mathrm{ABS}}$, $\mathcal{S}|\tilde{P}_i$ sends $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}, \mathit{ssid}, Q)$ to $\mathcal{F}_{\mathrm{AOS}}$. Then, upon receiving $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}, \mathit{ssid}, Q)$ from $\mathcal{F}_{\mathrm{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, \mathit{ack})$ to $\mathcal{F}_{\mathrm{AOS}}$.
Then, if $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no})$ from $\mathcal{F}_{\mathrm{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no}, \mathit{ack})$ to $\mathcal{F}_{\mathrm{AOS}}$. $\mathcal{S}|B$ then sends $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{no})$ to $\mathcal{S}|\mathcal{F}_{\mathrm{ABS}}$. Otherwise, if $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{yes})$ from $\mathcal{F}_{\mathrm{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_2, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\mathrm{AOS}}$. $\mathcal{S}|B$ then sends $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{yes})$ to $\mathcal{S}|\mathcal{F}_{\mathrm{ABS}}$.
If $\mathcal{S}|Q$ receives $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS}.\textbf{Response}, \sigma))$ from $\mathcal{S}|\mathcal{F}_{\mathrm{ASC}}$, and $\mathcal{S}|Q$ earlier sent $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS}.\textbf{Challenge}, c))$ to $\mathcal{S}|\mathcal{F}_{\mathrm{ASC}}$, then $\mathcal{S}|Q$ sends $(\textbf{ABS}.\textbf{Verify}, \mathit{sid}_2, \mathit{pk}, c, \sigma)$ to $\mathcal{S}|\mathcal{F}_{\mathrm{ABS}}$. Then, if $\mathcal{S}|Q$ receives $(\textbf{ABS}.\textbf{Verify}, \mathit{sid}_2, \mathit{accept})$ from $\mathcal{S}|\mathcal{F}_{\mathrm{ABS}}$, and $\mathcal{S}$ earlier received $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{yes})$ from $\mathcal{F}_{\mathrm{AOS}}$, then $\mathcal{S}$ stores (c, σ) and $(Q, \mathit{ssid}, \mathit{active})$, and sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{ack})$ to $\mathcal{F}_{\mathrm{AOS}}$.

Simulating the data exchange: When $\mathcal{S}|Q$ receives $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, m)$ from $\mathcal{S}|\mathcal{F}_{\mathrm{ASC}}$, if $(Q, \mathit{ssid}, \mathit{active})$ is stored, then $\mathcal{S}|\tilde{P}_i$ sends $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, m)$ to $\mathcal{F}_{\mathrm{AOS}}$, and upon receiving $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner}, Q, m)$ from $\mathcal{F}_{\mathrm{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner}, Q, \mathit{ack})$ to $\mathcal{F}_{\mathrm{AOS}}$.
When $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{partner}, m)$ from $\mathcal{F}_{\mathrm{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\mathrm{AOS}}$. If $(Q, \mathit{ssid}, \mathit{active})$ is stored, $\mathcal{S}|Q$ sends $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, m)$ to $\mathcal{S}|\mathcal{F}_{\mathrm{ASC}}$.

Simulating the session ending: If $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{ExpireSession}, Q, \mathit{sid}_1, \mathit{ssid})$ from $\mathcal{F}_{\mathrm{AOS}}$, then $\mathcal{S}|Q$ sends $(\textbf{ASC}.\textbf{ExpireSession}, \mathit{sid}_5, \mathit{ssid})$ to $\mathcal{S}|\mathcal{F}_{\mathrm{ASC}}$.

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 π_{AOS}, c is randomly chosen by an honest Q, while in $\text{IDEAL}_{\mathcal{F}_{\text{AOS}}}$, it is randomly chosen by $\mathcal{S}$, hence the distribution of c is the same in both cases.

We also observe the following difference between π_{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 $(\textbf{ABS.Verify}, \textit{sid}_2, \textit{pk}', c, \sigma)$ sent from Q to $\mathcal{F}_{\text{ABS}}$ has the correct key, i.e. $\textit{pk}' = \textit{pk}$, 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 $(c, \sigma, \textit{pk}, \textit{accept})$. Hence, by the non-forgeability condition, the signature will be rejected by $\mathcal{F}_{\text{ABS}}$.

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

When Q is corrupt, while P_i and B are honest: $\mathcal{S}$ 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 $\mathcal{S}$ obtains all the information needed to perfectly simulate the above functionalities.

Simulating the set-up: Upon receiving $(\textbf{AOS.Register}, \textit{sid}_1)$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}|B$ sends $(\textbf{ABS.KeyGen}, \textit{sid}_2)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. Upon receiving $(\textbf{ABS.Key}, \textit{sid}_2, \textit{pk})$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}|B$ sends $(\textbf{ACA.Register}, \textit{sid}_3, \textit{pk})$ to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$. Upon receiving $(\textbf{ACA.Registered}, \textit{sid}_3)$ from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}|B$ sends $(\textbf{SC.Register}, \textit{sid}_4)$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$. Upon receiving $(\textbf{SC.Registered}, \textit{sid}_4)$ from $\mathcal{S}|\mathcal{F}_{\text{SC}}$, $\mathcal{S}$ sends $(\textbf{AOS.Register}, \textit{sid}_1, \textit{ack})$ to $\mathcal{F}_{\text{AOS}}$.
The first time $\mathcal{S}|\tilde{Q}$ receives some input, $\mathcal{S}|\tilde{Q}$ sends $(\textbf{AOS.Register}, \textit{sid}_1)$ to $\mathcal{F}_{\text{ASC}}$, and upon receiving $(\textbf{AOS.Register}, \textit{sid}_1, Q)$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS.Register}, \textit{sid}_1, Q, \textit{ack})$ to $\mathcal{F}_{\text{AOS}}$.

Simulating the session set-up: Upon receiving $(\textbf{AOS.EstablishSession}, \textit{sid}_1, \textit{ssid}, Q)$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ chooses a random identity $\textit{ID}$ and stores $(\textit{ID}, \textit{ssid})$. $\mathcal{S}|\textit{ID}$ sends $(\textbf{ASC.EstablishSession}, \textit{sid}_5, \textit{ssid}, Q)$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\textbf{ASC.SessionEstablished}, \textit{sid}_5, \textit{ssid}, Q)$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|\textit{ID}$ sends $(\textbf{ACA.Retrieve}, \textit{sid}_3, B)$ to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$. Upon receiving $(\textbf{ACA.Retrieved}, \textit{sid}_3, B, \textit{pk})$ from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}|\textit{ID}$ sends $(\textbf{ASC.Send}, \textit{sid}_5, \textit{ssid}, \textbf{AOS.Pay})$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$.

If $\mathcal{S}|ID$ receives $(\textbf{ASC}.\textbf{Receive}, sid_5, ssid, (\textbf{AOS}.\textbf{Challenge}, c))$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|ID$ sends $(\textbf{ABS}.\textbf{Sign}, sid_2, ssid, pk, c)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. When $\mathcal{S}|B$ receives $(\textbf{ABS}.\textbf{Sign}, sid_2, ssid, ID, request)$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{EstablishSession}, sid_1, ssid, ack)$ to $\mathcal{F}_{\text{AOS}}$.
If $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{Pay}, sid_1, ssid, no)$ from $\mathcal{F}_{\text{AOS}}$, then $\mathcal{S}|B$ sends $(\textbf{ABS}.\textbf{Sign}, sid_2, ssid, no)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. Then, if $\mathcal{S}|ID$ receives $(\textbf{ABS}.\textbf{Sign}, sid_2, ssid, no)$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Pay}, sid_1, ssid, no, ack)$ to $\mathcal{F}_{\text{AOS}}$. Otherwise, if $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{Pay}, sid_1, ssid, yes)$ from $\mathcal{F}_{\text{AOS}}$, then $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Pay}, sid_1, ssid, Q, ack)$ to $\mathcal{F}_{\text{AOS}}$. $\mathcal{S}|B$ then sends $(\textbf{ABS}.\textbf{Sign}, sid_2, ssid, yes)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$.
If $\mathcal{S}|ID$ receives $(\textbf{ABS}.\textbf{Signature}, sid_2, ssid, \sigma)$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, then $\mathcal{S}|ID$ sends $(\textbf{ASC}.\textbf{Send}, sid_5, ssid, (\textbf{AOS}.\textbf{Response}, \sigma))$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. $\mathcal{S}$ stores $(ID, ssid, active)$ and sends $(\textbf{AOS}.\textbf{Pay}, sid_2, ssid, partner, ack)$ to $\mathcal{F}_{\text{AOS}}$.

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

Simulating the session ending: If $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{ExpireSession}, partner, sid_1, ssid)$ from $\mathcal{F}_{\text{AOS}}$, then $\mathcal{S}|ID$ sends $(\textbf{ASC}.\textbf{ExpireSession}, sid_5, ssid)$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$.

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

It can be verified that the simulation is perfect also in this case. The simulation guarantees that the number of pairs (c, σ) output by $\mathcal{S}|\mathcal{F}_{\text{ABS}}$ in the ideal protocol equals the number of such pairs output by $\mathcal{F}_{\text{ABS}}$ 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, σ), where $n \geq n'$. Recall that B is honest, so we can assume that every verification request $(\textbf{ABS.Verify}, \mathit{sid}_2, \mathit{pk}', c, \sigma)$ sent from B to $\mathcal{F}_{\text{ABS}}$ has the correct key, i.e. $\mathit{pk}' = \mathit{pk}$. In this case, the signature will be accepted only if $(c, \sigma, \mathit{pk}', \mathit{accept})$ is stored, which is the case only if (c, σ) was earlier output by $\mathcal{S}|\mathcal{F}_{\text{ABS}}/\mathcal{F}_{\text{ABS}}$. Since by assumption only n' pairs were output, at least one of the pairs will be rejected by $\mathcal{S}|\mathcal{F}_{\text{ABS}}/\mathcal{F}_{\text{ABS}}$ 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, σ), such that there is a recorded entry (c, σ') for some σ'. 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: $\mathcal{S}$ 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 $\mathcal{S}$ obtains all the information needed to perfectly simulate the above functionalities.

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

Simulating the session set-up: Upon receiving $(\textbf{AOS.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i, Q)$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$. $\mathcal{S}|P_i$ sends $(\textbf{ASC.EstablishSession}, \mathit{sid}_5, \mathit{ssid}, Q)$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\textbf{ASC.SessionEstablished}, \mathit{sid}_5, \mathit{ssid}, Q)$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|P_i$ sends $(\textbf{ACA.Retrieve}, \mathit{sid}_3, B)$ to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$. Upon receiving $(\textbf{ACA.Retrieved}, \mathit{sid}_3, B, \mathit{pk})$ from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}|P_i$ sends $(\textbf{ASC.Send}, \mathit{sid}_5, \mathit{ssid}, \textbf{AOS.Pay})$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\textbf{ASC.Receive}, \mathit{sid}_5, \mathit{ssid}, \textbf{AOS.Pay})$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}$ picks a random challenge c, and $\mathcal{S}|Q$ sends $(\textbf{ASC.Send}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS.Challenge}, c))$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\textbf{ASC.Receive}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS.Challenge}, c))$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|P_i$ sends $(\textbf{ABS.Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{pk}, c)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$.
If $\mathcal{S}|\mathcal{F}_{\text{ABS}}$ then receives $(\textbf{ABS.Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{no})$ from $A|B$, then $\mathcal{S}|\tilde{B}$ sends $(\textbf{AOS.Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no})$ to $\mathcal{F}_{\text{AOS}}$. When $\mathcal{S}|P_i$ receives $(\textbf{ABS.Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{no})$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}$ sends $(\textbf{AOS.Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.
Otherwise, if $\mathcal{S}|\mathcal{F}_{\text{ABS}}$ receives $(\textbf{ABS.Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{yes})$ from $A|B$, then $\mathcal{S}|\tilde{B}$ sends $(\textbf{AOS.Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{yes})$ to $\mathcal{F}_{\text{AOS}}$.
If $\mathcal{S}|P_i$ then receives $(\textbf{ABS.Signature}, \mathit{sid}_2, \mathit{ssid}, \sigma)$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, then $\mathcal{S}|P_i$ sends $(\textbf{ASC.Send}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS.Response}, \sigma))$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. $\mathcal{S}$ stores $(P_i, \mathit{ssid}, \mathit{active})$ and sends $(\textbf{AOS.Pay}, \mathit{sid}_2, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.

When $\mathcal{S}|Q$ receives $(\textbf{ASC.Receive}, sid_5, ssid, (\textbf{AOS.Response}, \sigma))$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|Q$ sends $(\textbf{ABS.Verify}, sid_2, pk, c, \sigma)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. If $\mathcal{S}|Q$ receives $(\textbf{ABS.Verify}, sid_2, accept)$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, then $\mathcal{S}$ stores (c, σ) and $(Q, ssid, active)$, and sends $(\textbf{AOS.Pay}, sid_1, ssid, Q, ack)$ 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 $(\textbf{AOS.Collect}, sid_1, ssid', Q, n)$ from $\mathcal{F}_{\text{AOS}}$, if $\mathcal{S}$ has at least n stored pairs (c, σ), then $\mathcal{S}|Q$ sends $(\textbf{SC.EstablishSession}, sid_4, ssid', B)$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$. When $\mathcal{S}|Q$ receives $(\textbf{SC.SessionEstablished}, sid_4, ssid', B)$ from $\mathcal{S}|\mathcal{F}_{\text{SC}}$, $\mathcal{S}$ lets s be a set of n stored pairs (c, σ), and sends $(\textbf{SC.Send}, sid_4, ssid', (\textbf{AOS.Collect}, s))$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$.
If $\mathcal{S}|Q$ receives $(\textbf{SC.Receive}, sid_4, ssid', \textbf{AOS.Collected})$ from $\mathcal{S}|\mathcal{F}_{\text{SC}}$, $\mathcal{S}$ removes the pairs (c, σ) contained in s, and $\mathcal{S}|Q$ sends $(\textbf{SC.ExpireSession}, sid_4, ssid')$ to $\mathcal{S}|\mathcal{F}_{\text{SC}}$. $\mathcal{S}$ sends $(\textbf{AOS.Collect}, sid_1, ssid', ack)$ to $\mathcal{F}_{\text{AOS}}$, and then sends $(\textbf{AOS.Collect}, sid_1, ssid', Q, ack)$ 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, $\mathcal{A}$ gets the challenge c from $\mathcal{F}_{\text{ABS}}$. In π_{AOS}, c is randomly chosen by an honest Q, while in $\text{IDEAL}_{\mathcal{F}_{\text{AOS}}}$, it is randomly chosen by $\mathcal{S}$, 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 $\mathcal{S}$ upon signature requests.

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

When P_i and Q are corrupt, while B is honest: $\mathcal{S}$ 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 $\mathcal{S}$ 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 $\mathcal{S}|\mathcal{F}_{\text{ABS}}$ sends $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, P_i, \mathit{request})$ to $\mathcal{S}|B$, then $\mathcal{S}|\tilde{P}_i$ sends $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, Q)$ to $\mathcal{F}_{\text{AOS}}$, and upon receiving $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, Q)$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.
If $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no})$ from $\mathcal{F}_{\text{AOS}}$, then $\mathcal{S}|B$ sends $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{no})$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. $\mathcal{S}$ then sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$. Otherwise, if $\mathcal{S}$ receives $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{yes})$ from $\mathcal{F}_{\text{AOS}}$, then $\mathcal{S}|B$ sends $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{yes})$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. $\mathcal{S}$ then sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$, and finally, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.

Simulating the data exchange and the session ending: Since both P_i and Q are corrupt, S obtains the inputs of both $\tilde{P}_i$ and $\tilde{Q}$, and also controls their outputs. $\mathcal{S}$ simulates honest B and 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.

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: $\mathcal{S}$ simulates honest Q and functionalities $\mathcal{F}_{\text{ABS}}$, $\mathcal{F}_{\text{ACA}}$, $\mathcal{F}_{\text{SC}}$ and $\mathcal{F}_{\text{ASC}}$. We observe that $\mathcal{S}$ 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 $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, \textbf{AOS}.\textbf{Pay})$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}$ picks a random challenge c, and $\mathcal{S}|Q$ sends $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS}.\textbf{Challenge}, c))$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$.
If $\mathcal{S}|Q$ receives $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS}.\textbf{Response}, \sigma))$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, and $\mathcal{S}|Q$ earlier sent $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS}.\textbf{Challenge}, c))$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, then $\mathcal{S}|Q$ sends $(\textbf{ABS}.\textbf{Verify}, \mathit{sid}_2, pk, c, \sigma)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$. Then, if $\mathcal{S}|Q$ receives $(\textbf{ABS}.\textbf{Verify}, \mathit{sid}_2, \mathit{accept})$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}$ proceeds as follows: $\mathcal{S}|\tilde{P}_i$ sends $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i, Q)$ to $\mathcal{F}_{\text{AOS}}$, and then $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$. $\mathcal{S}|\tilde{B}$ then sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{yes})$ to $\mathcal{F}_{\text{AOS}}$. $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_2, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$, and then $\mathcal{S}$ stores (c, σ) and $(Q, \mathit{ssid}, \mathit{active})$, and sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.

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

Simulating the set-up: The first time $\mathcal{S}|\tilde{Q}$ receives some input, $\mathcal{S}|\tilde{Q}$ sends $(\textbf{AOS}.\textbf{Register}, \mathit{sid}_1)$ to $\mathcal{F}_{\text{AOS}}$, and upon receiving $(\textbf{AOS}.\textbf{Register}, \mathit{sid}_1, Q)$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Register}, \mathit{sid}_1, Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.

Simulating the session set-up: Upon receiving $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i, Q)$ from $\mathcal{F}_{\text{AOS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$. $\mathcal{S}|P_i$ then sends $(\textbf{ASC}.\textbf{EstablishSession}, \mathit{sid}_5, \mathit{ssid}, Q)$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. Upon receiving $(\textbf{ASC}.\textbf{SessionEstablished}, \mathit{sid}_5, \mathit{ssid}, Q)$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|P_i$ sends $(\textbf{ACA}.\textbf{Retrieve}, \mathit{sid}_3, B)$ to $\mathcal{S}|\mathcal{F}_{\text{ACA}}$. Upon receiving $(\textbf{ACA}.\textbf{Retrieved}, \mathit{sid}_3, B, \mathit{pk})$ from $\mathcal{S}|\mathcal{F}_{\text{ACA}}$, $\mathcal{S}|P_i$ sends $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, \textbf{AOS}.\textbf{Pay})$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$.
If $\mathcal{S}|P_i$ receives $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS}.\textbf{Challenge}, c))$ from $\mathcal{S}|\mathcal{F}_{\text{ASC}}$, $\mathcal{S}|P_i$ sends $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{pk}, c)$ to $\mathcal{S}|\mathcal{F}_{\text{ABS}}$.
If $\mathcal{S}|\mathcal{F}_{\text{ABS}}$ then receives $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{no})$ from $A|B$, then $\mathcal{S}|\tilde{B}$ sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no})$ to $\mathcal{F}_{\text{AOS}}$. When $\mathcal{S}|P_i$ receives $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{no})$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.
Otherwise, if $\mathcal{S}|\mathcal{F}_{\text{ABS}}$ receives $(\textbf{ABS}.\textbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{yes})$ from $A|B$, then $\mathcal{S}$ sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, Q, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$. $\mathcal{S}|\tilde{B}$ then sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{yes})$ to $\mathcal{F}_{\text{AOS}}$.
If $\mathcal{S}|P_i$ then receives $(\textbf{ABS}.\textbf{Signature}, \mathit{sid}_2, \mathit{ssid}, \sigma)$ from $\mathcal{S}|\mathcal{F}_{\text{ABS}}$, then $\mathcal{S}|P_i$ sends $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, (\textbf{AOS}.\textbf{Response}, \sigma))$ to $\mathcal{S}|\mathcal{F}_{\text{ASC}}$. $\mathcal{S}$ stores $(P_i, \mathit{ssid}, \mathit{active})$ and sends $(\textbf{AOS}.\textbf{Pay}, \mathit{sid}_2, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ to $\mathcal{F}_{\text{AOS}}$.

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 $\tilde{Q}$ and $\tilde{B}$, and also controls their outputs. $\mathcal{S}$ simulates honest P_i and 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.

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. $\mathcal{S}$ 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 π_{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

[1] Aslak Bakke Buan, Kristian Gjøsteen, and Lillian Kråkmo. Universally Composable Blind Signatures in the Plain Model. Cryptology ePrint Archive, Report 2006/405, 2006. Available at `http://eprint.iacr.org/2006/405`.

[2] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Available at `http://eprint.iacr.org/2000/067`.

[3] Ran Canetti and Hugo Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 337–351, London, UK, 2002. Springer-Verlag.

[4] Ran Canetti and Tal Rabin. Universal Composition with Joint State. In *CRYPTO*, pages 265–281, 2003.

[5] Marc Fischlin. Round-Optimal Composable Blind Signatures in the Common Reference String Model. In *Advances in Cryptology-Crypto 2006*. Springer-Verlag, 2006.

[6] Waka Nagao, Yoshifumi Manabe, and Tatsuaki Okamoto. A Universally Composable Secure Channel Based on the KEM-DEM Framework. In *TCC*, pages 426–444, 2005.

[7] Kristian Gjøsteen and Lillian Kråkmo. Round-Optimal Blind Signatures from Waters Signatures. In *ProvSec '08: Proceedings of the 2nd International Conference on Provable Security*, pages 112–126, Berlin, Heidelberg, 2008. Springer-Verlag.

$\mathcal{F}_{\mathrm{AOS}}$ proceeds as follows, with a bank $\tilde{B}$, a server $\tilde{Q}$, users $\tilde{P}_1, \ldots, \tilde{P}_n$ and an ideal adversary $\mathcal{S}$.

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

  On message $(\mathbf{AOS}.\mathbf{EstablishSession}, \mathit{sid}, \mathit{ssid}, Q)$ from $\tilde{P}_i$, where $\mathit{sid} = (B, \mathit{sid}')$ and ssid has not been used before:
  1. Send $(\mathbf{AOS}.\mathbf{EstablishSession}, \mathit{sid}, \mathit{ssid}, Q)$ to $\mathcal{S}$ if $\tilde{B}$ is honest, and $(\mathbf{AOS}.\mathbf{EstablishSession}, \mathit{sid}, \mathit{ssid}, P_i, Q)$ otherwise.
  2. Upon receipt of $(\mathbf{AOS}.\mathbf{EstablishSession}, \mathit{sid}, \mathit{ssid}, Q, \mathit{ack})$ from $\mathcal{S}$, if Q is stored, then send $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, P_i, \mathit{request})$ to $\tilde{B}$.
  3. Upon receipt of $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, P_i, \mathit{no})$ from $\tilde{B}$, send $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, \mathit{no})$ to $\mathcal{S}$. Wait for $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, \mathit{no}, \mathit{ack})$ from $\mathcal{S}$, send $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, \mathit{no})$ to $\tilde{P}_i$ and stop. Otherwise, upon receipt of $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, P_i, \mathit{yes})$ from $\tilde{B}$, send $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, \mathit{yes})$ to $\mathcal{S}$.
  4. Upon receipt of $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, \mathit{partner}, \mathit{ack})$ from $\mathcal{S}$, store $(P_i, \mathit{ssid}, \mathit{active})$ and send $(\mathbf{AOS}.\mathbf{SessionEstablished}, \mathit{sid}, \mathit{ssid}, Q)$ to $\tilde{P}_i$.
  5. Upon receipt of $(\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}, \mathit{ssid}, Q, \mathit{ack})$ from $\mathcal{S}$, increase $\mathit{balance}$ and store $(Q, \mathit{ssid}, \mathit{active})$. Send $(\mathbf{AOS}.\mathbf{SessionEstablished}, \mathit{sid}, \mathit{ssid})$ to $\tilde{Q}$.
- **Data Exchange**
  - On message $(\mathbf{AOS}.\mathbf{Send}, \mathit{sid}, \mathit{ssid}, m)$ from $\tilde{P}_i$: If $(P_i, \mathit{ssid}, \mathit{active})$ is stored, then send $(\mathbf{AOS}.\mathbf{Send}, \mathit{sid}, \mathit{ssid}, \mathit{partner}, Q, |m|)$ to $\mathcal{S}$ if both $\tilde{P}_i$ and $\tilde{Q}$ are honest, and $(\mathbf{AOS}.\mathbf{Send}, \mathit{sid}, \mathit{ssid}, \mathit{partner}, Q, m)$ otherwise. Wait for $(\mathbf{AOS}.\mathbf{Send}, \mathit{sid}, \mathit{ssid}, \mathit{partner}, Q, \mathit{ack})$ from $\mathcal{S}$. If $(Q, \mathit{ssid}, \mathit{active})$ is stored, then send $(\mathbf{AOS}.\mathbf{Receive}, \mathit{sid}, \mathit{ssid}, m)$ to $\tilde{Q}$.
  - On message $(\mathbf{AOS}.\mathbf{Send}, \mathit{sid}, \mathit{ssid}, m)$ from $\tilde{Q}$: If $(Q, \mathit{ssid}, \mathit{active})$ is stored, then send $(\mathbf{AOS}.\mathbf{Send}, \mathit{sid}, \mathit{ssid}, Q, \mathit{partner}, |m|)$ to $\mathcal{S}$ if both $\tilde{P}_i$ and $\tilde{Q}$ are honest, and $(\mathbf{AOS}.\mathbf{Send}, \mathit{sid}, \mathit{ssid}, Q, \mathit{partner}, m)$ otherwise. Wait for $(\mathbf{AOS}.\mathbf{Send}, \mathit{sid}, \mathit{ssid}, Q, \mathit{partner}, \mathit{ack})$ from $\mathcal{S}$. If $(P_i, \mathit{ssid}, \mathit{active})$ is stored, then send $(\mathbf{AOS}.\mathbf{Receive}, \mathit{sid}, \mathit{ssid}, m)$ to $\tilde{P}_i$.
- **Session Ending**

  On message $(\mathbf{AOS}.\mathbf{ExpireSession}, \mathit{sid}, \mathit{ssid})$ from $\tilde{P}_i/\tilde{Q}$: Remove $(P_i, \mathit{ssid}, \mathit{active})/(Q, \mathit{ssid}, \mathit{active})$, and send $(\mathbf{AOS}.\mathbf{ExpireSession}, \mathit{sid}, \mathit{ssid}, \mathit{partner}/Q)$ to $\mathcal{S}$.
- **Collect**

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

FIGURE 1. The anonymous online service functionality $\mathcal{F}_{\mathrm{AOS}}$.

$\mathcal{F}_{\text{ASC}}$ proceeds as follows, with users $\tilde{P}_1, \ldots, \tilde{P}_n$ and an ideal adversary $\mathcal{S}$.

- **Set-up**
  In the first activation, expect to receive ($\mathbf{ASC.Register}$, *sid*) from some party $\tilde{P}_i$. Send ($\mathbf{ASC.Register}$, *sid*, P_i) to $\mathcal{S}$. Wait to receive ($\mathbf{ASC.Register}$, *sid*, *ack*) from $\mathcal{S}$, store P_i, and send ($\mathbf{ASC.Registered}$, *sid*) to $\tilde{P}_i$.
- **Session Set-up**
  Upon receiving ($\mathbf{ASC.EstablishSession}$, *sid*, *ssid*, P_i) from some party $\tilde{P}_j$, where *ssid* has not been used before, send ($\mathbf{ASC.EstablishSession}$, *sid*, *ssid*, P_i) to $\mathcal{S}$, and check that P_i is stored. Upon receiving ($\mathbf{ASC.EstablishSession}$, *sid*, *ssid*, *partner*, *ack*) from $\mathcal{S}$, store (P_j, *ssid*, *active*), and send ($\mathbf{ASC.SessionEstablished}$, *sid*, *ssid*, P_i) to $\tilde{P}_j$. Upon receiving ($\mathbf{ASC.EstablishSession}$, *sid*, *ssid*, P_i, *ack*) from $\mathcal{S}$, store (P_i, *ssid*, *active*), and send ($\mathbf{ASC.SessionEstablished}$, *sid*, *ssid*) to $\tilde{P}_i$.
- **Data Exchange**
  - Upon receiving ($\mathbf{ASC.Send}$, *sid*, *ssid*, m) from $\tilde{P}_i$, if (P_i, *ssid*, *active*) is stored, then send ($\mathbf{ASC.Send}$, *sid*, *ssid*, P_i, *partner*, $|m|$) to $\mathcal{S}$ if both P_i and P_j are honest, and ($\mathbf{ASC.Send}$, *sid*, *ssid*, P_i, *partner*, m) otherwise. Wait to receive ($\mathbf{ASC.Send}$, *sid*, *ssid*, P_i, *partner*, *ack*) from $\mathcal{S}$. If (P_j, *ssid*, *active*) is stored, then send ($\mathbf{ASC.Receive}$, *sid*, *ssid*, m) to $\tilde{P}_j$.
  - Upon receiving ($\mathbf{ASC.Send}$, *sid*, *ssid*, m') from $\tilde{P}_j$, if (P_j, *ssid*, *active*) is stored, then send ($\mathbf{ASC.Send}$, *sid*, *ssid*, *partner*, P_i, $|m'|$) to $\mathcal{S}$ if both P_i and P_j are honest, and ($\mathbf{ASC.Send}$, *sid*, *ssid*, *partner*, P_i, m') otherwise. Wait to receive ($\mathbf{ASC.Send}$, *sid*, *ssid*, *partner*, P_i, *ack*) from $\mathcal{S}$. If (P_i, *ssid*, *active*) is stored, then send ($\mathbf{ASC.Receive}$, *sid*, *ssid*, m') to $\tilde{P}_i$.
- **Session Ending**
  Upon receiving ($\mathbf{ASC.ExpireSession}$, *sid*, *ssid*) from $\tilde{P}_i/\tilde{P}_j$, remove (P_i, *ssid*, *active*)/(P_j, *ssid*, *active*), and send ($\mathbf{ASC.ExpireSession}$, *sid*, *ssid*, P_i/*partner*) to $\mathcal{S}$.

FIGURE 2. The anonymous secure channel functionality $\mathcal{F}_{\text{ASC}}$.

$\mathcal{F}_{\text{AN}}$ proceeds as follows, with users $\tilde{P}_1, \ldots, \tilde{P}_n$ and an ideal adversary $\mathcal{S}$.

- In the first activation, expect to receive ($\mathbf{AN.Register}$, *sid*) from some party $\tilde{P}_j$. Choose a random identity *ID*. Send ($\mathbf{AN.Registered}$, *sid*, *ID*) to $\tilde{P}_j$ and store (P_j, *ID*).
- Upon receiving ($\mathbf{AN.Send}$, *sid*, m, ID') from some party $\tilde{P}_i$, if $i = j$, then send ($\mathbf{AN.Send}$, *sid*, m, *ID*, ID') to $\mathcal{S}$, otherwise send ($\mathbf{AN.Send}$, *sid*, m, P_i, ID') to $\mathcal{S}$. Upon receiving ($\mathbf{AN.Send}$, *sid*, m', ID'') from $\mathcal{S}$, if $ID'' = ID$, then send ($\mathbf{AN.Receive}$, *sid*, m') to $\tilde{P}_j$, otherwise send ($\mathbf{AN.Receive}$, *sid*, m') to $\tilde{ID}''$.

FIGURE 3. The anonymous network functionality $\mathcal{F}_{\text{AN}}$.

$\mathcal{F}_{\mathrm{ACA}}$, proceeds as follows, with parties $\tilde{P}_1, \ldots, \tilde{P}_n$ and an ideal adversary $\mathcal{S}$.

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

Figure 4. The ideal anonymous certificate authority functionality $\mathcal{F}_{\mathrm{ACA}}$.

$\mathcal{F}_{\text{KEM-DEM}}$ proceeds as follows, with parties $\tilde{P}_1, \dots, \tilde{P}_n$ and an ideal adversary $\mathcal{S}$.

- **Set-up**

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

  Upon receiving (**KEM**.**Encrypt**, *sid*, *ssid*, pk') from some party $\tilde{P}_j$:
  - If there is no recorded entry (P_j, *ssid*, C, *active*) for any C, then send (**KEM**.**Encrypt**, *sid*, *ssid*, pk') to $\mathcal{S}$. Upon receiving (**EncryptedSharedKey**, *sid*, *ssid*, pk', C_0) from $\mathcal{S}$, send (**EncryptedSharedKey**, *sid*, *ssid*, pk', C_0) to $\tilde{P}_j$, and store (pk', *ssid*, C_0) and (P_j, *ssid*, C_0, *active*).
  - Otherwise, do nothing.

  Upon receiving (**KEM**.**Decrypt**, *sid*, *ssid*, C'_0) from $\tilde{P}_i$ (and $\tilde{P}_i$ only), send (**KEM**.**Decrypt**, *sid*, *ssid*, C'_0) to $\mathcal{S}$. Upon receiving (**KEM**.**Decrypt**, *sid*, *ssid*, *ack*) from $\mathcal{S}$:
  - If there is no stored entry (P_i, *ssid*, C, *active*) for any C, send (**KEM**.**Decrypt**, *sid*, *ssid*, *ok*) to $\tilde{P}_i$ and store (P_i, *ssid*, C'_0, *active*).
- **Data Exchange**

  Upon receiving (**DEM**.**Encrypt**, *sid*, *ssid*, m) from $\tilde{P}_i/\tilde{P}_j$:
  - If (P_i/P_j, *ssid*, C_0, *active*) is stored:
    - If both P_i and P_j are honest, then send (**DEM**.**Encrypt**, *sid*, *ssid*, $|m|$) to $\mathcal{S}$. Upon receiving (**DEM**.**Ciphertext**, *sid*, *ssid*, c) from $\mathcal{S}$, send (**DEM**.**Ciphertext**, *sid*, *ssid*, c) to $\tilde{P}_i/\tilde{P}_j$, and store (*ssid*, m, c, C_0).
    - Otherwise, send (**DEM**.**Encrypt**, *sid*, *ssid*, m) to $\mathcal{S}$. Upon receiving (**DEM**.**Ciphertext**, *sid*, *ssid*, c) from $\mathcal{S}$, send (**DEM**.**Ciphertext**, *sid*, *ssid*, c) to $\tilde{P}_i/\tilde{P}_j$, and store (*ssid*, m, c, C_0).

  Upon receiving (**DEM**.**Decrypt**, *sid*, *ssid*, c') from $\tilde{P}_i/\tilde{P}_j$, send (**DEM**.**Decrypt**, *sid*, *ssid*, c') to $\mathcal{S}$. Upon receiving (**DEM**.**Plaintext**, *sid*, *ssid*, m') from $\mathcal{S}$:
  - If there is a recorded entry (P_i/P_j, *ssid*, C, *active*) for some C:
    1. If there is a recorded entry (*ssid*, m, c', C), then send (**DEM**.**Plaintext**, *sid*, *ssid*, m) to $\tilde{P}_i/\tilde{P}_j$.
    2. Otherwise, if both P_i and P_j are honest, then store (*ssid*, $\perp$, c', C) and send (**DEM**.**Plaintext**, *sid*, *ssid*, $\perp$) to $\tilde{P}_i/\tilde{P}_j$.
    3. Otherwise, if there is a recorded entry (*ssid*, $\perp$, c', C), then send (**DEM**.**Plaintext**, *sid*, *ssid*, $\perp$) to $\tilde{P}_i/\tilde{P}_j$.
    4. Otherwise, send (**DEM**.**Plaintext**, *sid*, *ssid*, m') to $\tilde{P}_i/\tilde{P}_j$, and store (*ssid*, m', c', C).
- **Session Ending**

  Upon receiving (**KEM**.**ExpireSession**, *sid*, *ssid*) from $\tilde{P}_i/\tilde{P}_j$, remove (P_i, *ssid*, C_0, *active*)/ (P_j, *ssid*, C'_0, *active*).

FIGURE 5. The KEM-DEM functionality $\mathcal{F}_{\text{KEM-DEM}}$.

π_{ASC} runs as follows, with users $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$.

- **Set-up**

  Upon the first input $(\textbf{ASC.Register}, \mathit{sid}_1)$, P_i sends $(\textbf{KEM.KeyGen}, \mathit{sid}_2)$ to $\mathcal{F}_{\text{KEM-DEM}}$. Upon receiving $(\textbf{KEM.Key}, \mathit{sid}_2, PK_i)$ from $\mathcal{F}_{\text{KEM-DEM}}$, P_i sends $(\textbf{ACA.Register}, \mathit{sid}_3, PK_i)$ to $\mathcal{F}_{\text{ACA}}$. Upon receiving $(\textbf{ACA.Registered}, \mathit{sid}_3)$ from $\mathcal{F}_{\text{ACA}}$, P_i outputs $(\textbf{ASC.Registered}, \mathit{sid}_1)$.
- **Session Set-up**
  1. Upon the first input $(\textbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, P_i)$, P_j sends $(\textbf{AN.Register}, (\mathit{sid}_4, \mathit{ssid}))$ to $\mathcal{F}_{\text{AN}}$, and waits to receive $(\textbf{AN.Registered}, (\mathit{sid}_4, \mathit{ssid}), \mathit{ID})$ from $\mathcal{F}_{\text{AN}}$. Then P_j sends $(\textbf{ACA.Retrieve}, \mathit{sid}_3, P_i)$ to $\mathcal{F}_{\text{ACA}}$.
  2. Upon receiving $(\textbf{ACA.Retrieved}, \mathit{sid}_3, P_i, PK_i)$ from $\mathcal{F}_{\text{ACA}}$, P_j sends $(\textbf{KEM.Encrypt}, \mathit{sid}_2, \mathit{ssid}, PK_i)$ to $\mathcal{F}_{\text{KEM-DEM}}$, and receives $(\textbf{EncryptedSharedKey}, \mathit{sid}_2, \mathit{ssid}, PK_i, C_0)$ from $\mathcal{F}_{\text{KEM-DEM}}$.
  3. P_j stores $(\mathit{ssid}, \mathit{active})$, sets variables $\mathit{ssid} - \mathit{sn}$ and $\mathit{ssid} - \mathit{psn}$ equal to 1, sends $(\textbf{AN.Send}, (\mathit{sid}_4, \mathit{ssid}), (\textbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, C_0, \mathit{ID}), P_i)$ to $\mathcal{F}_{\text{AN}}$, and outputs $(\textbf{ASC.SessionEstablished}, \mathit{sid}_1, \mathit{ssid}, P_i)$.
  4. Upon receiving $(\textbf{AN.Receive}, (\mathit{sid}_4, \mathit{ssid}), (\textbf{ASC.EstablishSession}, \mathit{sid}_1, \mathit{ssid}, C_0', \mathit{ID}'))$ from $\mathcal{F}_{\text{AN}}$, P_i sends $(\textbf{KEM.Decrypt}, \mathit{sid}_2, \mathit{ssid}, C_0')$ to $\mathcal{F}_{\text{KEM-DEM}}$. If $(\textbf{KEM.Decrypt}, \mathit{sid}_2, \mathit{ssid}, \mathit{ok})$ is returned from $\mathcal{F}_{\text{KEM-DEM}}$, P_i stores $(\mathit{ssid}, \mathit{active})$, sets variables $\mathit{ssid} - \mathit{sn}$ and $\mathit{ssid} - \mathit{psn}$ equal to 1 and outputs $(\textbf{ASC.SessionEstablished}, \mathit{sid}_1, \mathit{ssid})$.
- **Data Exchange**
  1. Upon input $(\textbf{ASC.Send}, \mathit{sid}_1, \mathit{ssid}, M)$, P_i/P_j checks that $(\mathit{ssid}, \mathit{active})$ is stored, lets $m = M||P_i/\mathit{ID}||\mathit{ssid} - \mathit{sn}$, increases $\mathit{ssid} - \mathit{sn}$ and sends $(\textbf{DEM.Encrypt}, \mathit{sid}_2, \mathit{ssid}, m)$ to $\mathcal{F}_{\text{KEM-DEM}}$.
  2. Upon receiving $(\textbf{DEM.Ciphertext}, \mathit{sid}_2, \mathit{ssid}, c)$ from $\mathcal{F}_{\text{KEM-DEM}}$, P_i/P_j sends $(\textbf{AN.Send}, (\mathit{sid}_4, \mathit{ssid}), c, \mathit{ID}/P_i)$ to $\mathcal{F}_{\text{AN}}$.
  3. Upon receiving $(\textbf{AN.Receive}, (\mathit{sid}_4, \mathit{ssid}), c)$ from $\mathcal{F}_{\text{AN}}$, P_j/P_i checks that $(\mathit{ssid}, \mathit{active})$ is stored, and sends $(\textbf{DEM.Decrypt}, \mathit{sid}_2, \mathit{ssid}, c)$ to $\mathcal{F}_{\text{KEM-DEM}}$.
  4. Upon receiving $(\textbf{DEM.Plaintext}, \mathit{sid}_2, \mathit{ssid}, m)$ from $\mathcal{F}_{\text{KEM-DEM}}$, P_j/P_i checks that $m = M||P_i/\mathit{ID}||\mathit{ssid} - \mathit{psn}$, increases $\mathit{ssid} - \mathit{psn}$ and outputs $(\textbf{ASC.Receive}, \mathit{sid}_1, \mathit{ssid}, M)$.
- **Session Ending**

  Upon input $(\textbf{ASC.ExpireSession}, \mathit{sid}_1, \mathit{ssid})$, P_i/P_j removes $(\mathit{ssid}, \mathit{active})$ and sends $(\textbf{KEM.ExpireSession}, \mathit{sid}_2, \mathit{ssid})$ to $\mathcal{F}_{\text{KEM-DEM}}$.

FIGURE 6. The anonymous secure channel protocol π_{ASC}.

$\mathcal{F}_{\text{ABS}}$ proceeds as follows, with signer $\tilde{B}$, users $\tilde{P}_1, \ldots, \tilde{P}_n$ and an ideal adversary $\mathcal{S}$.

- Upon receiving the first (**ABS.KeyGen**, *sid*) from the signer $\tilde{B}$, where $sid = (B, sid')$, send (**ABS.KeyGen**, *sid*) to $\mathcal{S}$. Upon receiving (**ABS.Key**, sid, pk, Π) from $\mathcal{S}$, store (pk, Π), send (**ABS.Key**, sid, pk) to $\tilde{B}$ and stop.
- Upon receiving (**ABS.Sign**, $sid, ssid, pk', m$) from a user $\tilde{P}_i$, where $sid = (B, sid')$, proceed as follows:
  1. If $pk' = pk$ and $\tilde{P}_i$ is honest, then send (**ABS.Sign**, $sid, ssid, pk'$) to $\mathcal{S}$, otherwise send (**ABS.Sign**, $sid, ssid, pk', m$) to $\mathcal{S}$.
  2. Upon receiving (**ABS.Sign**, $sid, ssid, pk', ack$) from $\mathcal{S}$, send (**ABS.Sign**, $sid, ssid, P_i, request$) to $\tilde{B}$.
  3. Upon receiving (**ABS.Sign**, $sid, ssid, P_i, no$) from $\tilde{B}$, send (**ABS.Sign**, $sid, ssid, no$) to $\mathcal{S}$. Wait for (**ABS.Sign**, $sid, ssid, no, ack$) from $\mathcal{S}$, and send (**ABS.Sign**, $sid, ssid, no$) to $\tilde{P}_i$.
  4. Upon receiving (**ABS.Sign**, $sid, ssid, P_i, yes$) from $\tilde{B}$, send (**ABS.Sign**, $sid, ssid, yes$) to $\mathcal{S}$.
     (a) Upon receiving (**ABS.Signature**, $sid, ssid$, *signer completed*) from $\mathcal{S}$, send (**ABS.Signature**, $sid, ssid, P_i$) to $\tilde{B}$.
     (b) Upon receiving (**ABS.Signature**, $sid, ssid$, *signer not completed*) from $\mathcal{S}$, send (**ABS.Signature**, $sid, ssid, P_i$, *signer not completed*) to $\tilde{B}$.
     (c) Upon receiving (**ABS.Signature**, $sid, ssid$, *user completed*, σ') from $\mathcal{S}$, proceed as follows: If $\tilde{P}_i$ is honest and $pk' = pk$, then generate $\sigma \overset{r}{\leftarrow} \Pi(m)$. If $(m, \sigma, pk', reject)$ is stored, then stop, otherwise store $(m, \sigma, pk', accept)$ and send (**ABS.Signature**, $sid, ssid, \sigma$) to $\tilde{P}_i$. Otherwise, if $\tilde{P}_i$ is corrupt or $pk' \neq pk$, and if there is no stored entry $(m, \sigma', pk', reject)$, then store $(m, \sigma', pk', accept)$ and send (**ABS.Signature**, $sid, ssid, \sigma'$) to $\tilde{P}_i$.
     (d) Upon receiving (**ABS.Signature**, $sid, ssid$, *user failed*) from $\mathcal{S}$, send (**ABS.Signature**, $sid, ssid$, *user failed*) to $\tilde{P}_i$.
- Upon receiving (**ABS.Verify**, sid, pk', m, σ) from an honest user $\tilde{P}_j$, send (**ABS.Verify**, sid, pk', m, σ) to $\mathcal{S}$. Upon receiving (**ABS.Verify**, $sid, pk', m, \sigma, \phi$) from $\mathcal{S}$, proceed as follows:
  1. If $pk' = pk$ and $(m, \sigma, pk, accept)$ is stored, then let $f = accept$ (completeness condition).
  2. Otherwise, if $pk' = pk$, B is honest and there is no recorded entry $(m, \sigma, pk, accept)$, then let $f = reject$ and store $(m, \sigma, pk, reject)$ (non-forgeability condition).
  3. Otherwise, if there is a recorded entry (m, σ, pk', f'), then let $f = f'$ (consistency condition).
  4. Otherwise, let $f = \phi$ and store (m, σ, pk', f).

  Send (**ABS.Verify**, sid, f) to $\tilde{P}_j$.

FIGURE 7. The anonymous blind signature functionality $\mathcal{F}_{\text{ABS}}$.

$\mathcal{F}_{\text{SC}}$ proceeds as follows, with users $\tilde{P}_1, \dots, \tilde{P}_n$ and an ideal adversary $\mathcal{S}$.

- **Set-up**
  In the first activation, expect to receive (**SC.Register**, *sid*) from some party $\tilde{P}_i$. Send (**SC.Register**, *sid*, P_i) to $\mathcal{S}$. Wait to receive (**SC.Register**, *sid*, *ack*) from $\mathcal{S}$, store P_i, and send (**SC.Registered**, *sid*) to $\tilde{P}_i$.
- **Session Set-up**
  Upon receiving (**SC.EstablishSession**, *sid*, *ssid*, P_i) from some party $\tilde{P}_j$, send (**SC.EstablishSession**, *sid*, *ssid*, P_i, P_j) to $\mathcal{S}$, and check that P_i is stored. Upon receiving (**SC.EstablishSession**, *sid*, *ssid*, P_j, *ack*) from $\mathcal{S}$, store (P_j, *ssid*, *active*), and send (**SC.SessionEstablished**, *sid*, *ssid*, P_i) to $\tilde{P}_j$. Upon receiving (**SC.EstablishSession**, *sid*, *ssid*, P_i, *ack*) from $\mathcal{S}$, store (P_i, *ssid*, *active*), and send (**SC.SessionEstablished**, *sid*, *ssid*, P_j) to $\tilde{P}_i$.
- **Data Exchange**
  - Upon receiving (**SC.Send**, *sid*, *ssid*, m) from $\tilde{P}_i$, if (P_i, *ssid*, *active*) is stored, then send (**SC.Send**, *sid*, *ssid*, $P_i, P_j, |m|$) to $\mathcal{S}$ if both P_i and P_j are honest, and (**SC.Send**, *sid*, *ssid*, P_i, P_j, m) otherwise. Wait to receive (**SC.Send**, *sid*, *ssid*, P_i, P_j, *ack*) from $\mathcal{S}$. If (P_j, *ssid*, *active*) is stored, then send (**SC.Receive**, *sid*, *ssid*, m) to $\tilde{P}_j$.
  - Upon receiving (**SC.Send**, *sid*, *ssid*, m) from $\tilde{P}_j$, if (P_j, *ssid*, *active*) is stored, then send (**SC.Send**, *sid*, *ssid*, $P_j, P_i, |m|$) to $\mathcal{S}$ if both P_i and P_j are honest, and (**SC.Send**, *sid*, *ssid*, P_j, P_i, m) otherwise. Wait to receive (**SC.Send**, *sid*, *ssid*, P_j, P_i, *ack*) from $\mathcal{S}$. If (P_i, *ssid*, *active*) is stored, then send (**SC.Receive**, *sid*, *ssid*, m) to $\tilde{P}_i$.
- **Session Ending**
  Upon receiving (**SC.ExpireSession**, *sid*, *ssid*) from $\tilde{P}_i/\tilde{P}_j$, remove (P_i, *ssid*, *active*)/(P_j, *ssid*, *active*), and send (**SC.ExpireSession**, *sid*, *ssid*, P_i/P_j) to $\mathcal{S}$.

FIGURE 8. The secure channel functionality $\mathcal{F}_{\text{SC}}$.

π_{AOS} runs as follows, with a bank B, a server Q, users $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$.

- **Set-up**
  - Upon the first input ($\mathbf{AOS}.\mathbf{Register}, \mathit{sid}_1$), where $\mathit{sid}_1 = (B, \mathit{sid}')$, B sends ($\mathbf{ABS}.\mathbf{KeyGen}, \mathit{sid}_2$) to $\mathcal{F}_{\mathrm{ABS}}$. Upon receiving ($\mathbf{ABS}.\mathbf{Key}, \mathit{sid}_2, \mathit{pk}$) from $\mathcal{F}_{\mathrm{ABS}}$, B sends ($\mathbf{ACA}.\mathbf{Register}, \mathit{sid}_3, \mathit{pk}$) to $\mathcal{F}_{\mathrm{ACA}}$. Upon receiving ($\mathbf{ACA}.\mathbf{Registered}, \mathit{sid}_3$) from $\mathcal{F}_{\mathrm{ACA}}$, B sends ($\mathbf{SC}.\mathbf{Register}, \mathit{sid}_4$) to $\mathcal{F}_{\mathrm{SC}}$. Upon receiving ($\mathbf{SC}.\mathbf{Registered}, \mathit{sid}_4$) from $\mathcal{F}_{\mathrm{SC}}$, B outputs ($\mathbf{AOS}.\mathbf{Registered}, \mathit{sid}_1$).
  - Upon the first input ($\mathbf{AOS}.\mathbf{Register}, \mathit{sid}_1$), where $\mathit{sid}_1 = (B, \mathit{sid}')$, Q sends ($\mathbf{ACA}.\mathbf{Retrieve}, \mathit{sid}_3, B$) to $\mathcal{F}_{\mathrm{ACA}}$. Upon receiving ($\mathbf{ACA}.\mathbf{Retrieved}$, $\mathit{sid}_3, B, \mathit{pk}$) from $\mathcal{F}_{\mathrm{ACA}}$, Q sends ($\mathbf{ASC}.\mathbf{Register}, \mathit{sid}_5$) to $\mathcal{F}_{\mathrm{ASC}}$, and waits to receive ($\mathbf{ASC}.\mathbf{Registered}, \mathit{sid}_5$) from $\mathcal{F}_{\mathrm{ASC}}$.
- **Session Set-up**
  1. Upon input ($\mathbf{AOS}.\mathbf{EstablishSession}, \mathit{sid}_1, \mathit{ssid}, Q$), where $\mathit{sid}_1 = (B, \mathit{sid}')$, P_i sends ($\mathbf{ASC}.\mathbf{EstablishSession}, \mathit{sid}_5, \mathit{ssid}, Q$) to $\mathcal{F}_{\mathrm{ASC}}$. Upon receiving ($\mathbf{ASC}.\mathbf{SessionEstablished}, \mathit{sid}_5, \mathit{ssid}, Q$) from $\mathcal{F}_{\mathrm{ASC}}$, P_i sends ($\mathbf{ACA}.\mathbf{Retrieve}, \mathit{sid}_3, B$) to $\mathcal{F}_{\mathrm{ACA}}$. Upon receiving ($\mathbf{ACA}.\mathbf{Retrieved}$, $\mathit{sid}_3, B, \mathit{pk}$) from $\mathcal{F}_{\mathrm{ACA}}$, P_i sends ($\mathbf{ASC}.\mathbf{Send}, \mathit{sid}_5, \mathit{ssid}, \mathbf{AOS}.\mathbf{Pay}$) to $\mathcal{F}_{\mathrm{ASC}}$.
  2. Upon receiving ($\mathbf{ASC}.\mathbf{Receive}, \mathit{sid}_5, \mathit{ssid}, \mathbf{AOS}.\mathbf{Pay}$) from $\mathcal{F}_{\mathrm{ASC}}$, Q picks a random challenge c, and sends ($\mathbf{ASC}.\mathbf{Send}, \mathit{sid}_5, \mathit{ssid}, (\mathbf{AOS}.\mathbf{Challenge}, c)$) to $\mathcal{F}_{\mathrm{ASC}}$.
  3. Upon receiving ($\mathbf{ASC}.\mathbf{Receive}, \mathit{sid}_5, \mathit{ssid}, (\mathbf{AOS}.\mathbf{Challenge}, c)$) from $\mathcal{F}_{\mathrm{ASC}}$, P_i sends ($\mathbf{ABS}.\mathbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{pk}, c$) to $\mathcal{F}_{\mathrm{ABS}}$.
  4. Upon receiving ($\mathbf{ABS}.\mathbf{Sign}, \mathit{sid}_2, \mathit{ssid}, P_i, \mathit{request}$) from $\mathcal{F}_{\mathrm{ABS}}$, B outputs ($\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}_1, \mathit{ssid}, P_i, \mathit{request}$). B proceeds as follows:
     - Upon input ($\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no}$), B sends ($\mathbf{ABS}.\mathbf{Sign}$, $\mathit{sid}_2, \mathit{ssid}, \mathit{no}$) to $\mathcal{F}_{\mathrm{ABS}}$.
     - Upon input ($\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{yes}$), B sends ($\mathbf{ABS}.\mathbf{Sign}$, $\mathit{sid}_2, \mathit{ssid}, \mathit{yes}$) to $\mathcal{F}_{\mathrm{ABS}}$.
  5. P_i proceeds as follows:
     - Upon receiving ($\mathbf{ABS}.\mathbf{Sign}, \mathit{sid}_2, \mathit{ssid}, \mathit{no}$) from $\mathcal{F}_{\mathrm{ABS}}$, P_i outputs ($\mathbf{AOS}.\mathbf{Pay}, \mathit{sid}_1, \mathit{ssid}, \mathit{no}$).
     - Upon receiving ($\mathbf{ABS}.\mathbf{Signature}, \mathit{sid}_2, \mathit{ssid}, \sigma$) from $\mathcal{F}_{\mathrm{ABS}}$, P_i sends ($\mathbf{ASC}.\mathbf{Send}, \mathit{sid}_5, \mathit{ssid}, (\mathbf{AOS}.\mathbf{Response}, \sigma)$) to $\mathcal{F}_{\mathrm{ASC}}$, sets a boolean variable *active* and outputs ($\mathbf{AOS}.\mathbf{SessionEstablished}, \mathit{sid}_1, \mathit{ssid}, Q$).
  6. Upon receiving ($\mathbf{ASC}.\mathbf{Receive}, \mathit{sid}_5, \mathit{ssid}, (\mathbf{AOS}.\mathbf{Response}, \sigma)$) from $\mathcal{F}_{\mathrm{ASC}}$, Q sends ($\mathbf{ABS}.\mathbf{Verify}, \mathit{sid}_2, \mathit{pk}, c, \sigma$) to $\mathcal{F}_{\mathrm{ABS}}$. Upon receiving ($\mathbf{ABS}.\mathbf{Verify}$, $\mathit{sid}_2, \mathit{accept}$) from $\mathcal{F}_{\mathrm{ABS}}$, Q stores (c, σ), sets a boolean variable *active* and outputs ($\mathbf{AOS}.\mathbf{SessionEstablished}, \mathit{sid}_1, \mathit{ssid}$).

FIGURE 9. The anonymous online service protocol π_{AOS}, part 1.

π_{AOS} runs as follows, with a bank B, a server Q, users $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$.

- **Data Exchange**
  (1) On input $(\textbf{AOS}.\textbf{Send}, \mathit{sid}_1, \mathit{ssid}, m)$, if P_i/Q is *active*, then he sends $(\textbf{ASC}.\textbf{Send}, \mathit{sid}_5, \mathit{ssid}, m)$ to $\mathcal{F}_{\text{ASC}}$.
  (2) Upon receiving $(\textbf{ASC}.\textbf{Receive}, \mathit{sid}_5, \mathit{ssid}, m)$ from $\mathcal{F}_{\text{ASC}}$, if Q/P_i is *active*, then he outputs $(\textbf{AOS}.\textbf{Receive}, \mathit{sid}_1, \mathit{ssid}, m)$.
- **Session Ending**
  Upon input $(\textbf{AOS}.\textbf{ExpireSession}, \mathit{sid}_1, \mathit{ssid})$, P_i/Q unsets the *active* variable and sends $(\textbf{ASC}.\textbf{ExpireSession}, \mathit{sid}_5, \mathit{ssid})$ to $\mathcal{F}_{\text{ASC}}$.
- **Collect**
  (1) On input $(\textbf{AOS}.\textbf{Collect}, \mathit{sid}_1, \mathit{ssid}', n)$, if Q has at least n stored pairs (c, σ), Q sends $(\textbf{SC}.\textbf{EstablishSession}, \mathit{sid}_4, \mathit{ssid}', B)$ to $\mathcal{F}_{\text{SC}}$.
  (2) Upon receiving $(\textbf{SC}.\textbf{SessionEstablished}, \mathit{sid}_4, \mathit{ssid}', B)$ from $\mathcal{F}_{\text{SC}}$, Q lets s be a set of n pairs (c, σ), and sends $(\textbf{SC}.\textbf{Send}, \mathit{sid}_4, \mathit{ssid}', (\textbf{AOS}.\textbf{Collect}, s))$ to $\mathcal{F}_{\text{SC}}$.
  (3) Upon receiving $(\textbf{SC}.\textbf{Receive}, \mathit{sid}_4, \mathit{ssid}', (\textbf{AOS}.\textbf{Collect}, s))$ from $\mathcal{F}_{\text{SC}}$, for every pair (c, σ) in s, B checks that there is no stored pair (c, σ') for any σ', and sends $(\textbf{ABS}.\textbf{Verify}, \mathit{sid}_2, \mathit{pk}, c, \sigma)$ to $\mathcal{F}_{\text{ABS}}$. If, for every pair, B receives $(\textbf{ABS}.\textbf{Verify}, \mathit{sid}_2, \mathit{accept})$ from $\mathcal{F}_{\text{ABS}}$, B stores s and sends $(\textbf{SC}.\textbf{Send}, \mathit{sid}_4, \mathit{ssid}', \textbf{AOS}.\textbf{Collected})$ to $\mathcal{F}_{\text{SC}}$. B then sends $(\textbf{SC}.\textbf{ExpireSession}, \mathit{sid}_4, \mathit{ssid}')$ to $\mathcal{F}_{\text{SC}}$ and outputs $(\textbf{AOS}.\textbf{Deposit}, \mathit{sid}_1, \mathit{ssid}', Q, n)$.
  (4) Upon receiving $(\textbf{SC}.\textbf{Receive}, \mathit{sid}_4, \mathit{ssid}', \textbf{AOS}.\textbf{Collected})$ from $\mathcal{F}_{\text{SC}}$, Q removes the pairs (c, σ) corresponding to s, Q then sends $(\textbf{SC}.\textbf{ExpireSession}, \mathit{sid}_4, \mathit{ssid}')$ to $\mathcal{F}_{\text{SC}}$ and outputs $(\textbf{AOS}.\textbf{Collected}, \mathit{sid}_1, \mathit{ssid}', n)$.

FIGURE 10. The anonymous online service protocol π_{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 KRÅKMO

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* $\mathcal{Z}$ 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*, 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, $dom(f)_2$ 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 σ is a function and τ 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 σ is a bijection, τ restricted to $dom(f_1)_2$ is a bijection on $dom(f_2)_2$, $\sigma(W_1) = W_2$ and $f_1(s, w)$ is defined if and only if $f_2(\sigma(s), \tau(w))$ is defined, then the map μ 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 $w_1 w_2 \dots w_n \in I^*$ 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 τ to each letter, removing letters for which τ is undefined.

Lemma 1. *Let M_1 and M_2 be state machines. If (σ, τ) is a map from M_1 to M_2, then*

$$\tau(L(M_1)) \subseteq L(M_2).$$

If (σ, τ) 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 $w_1 w_2 \dots w_n \in I_1^*$. Suppose M_1 recognizes this word. Then we have states $s_{1,0}, s_{1,1}, \dots, s_{1,n} \in S_1$ 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 $\tau(w_1 w_2 \dots w_n)$, and $\tau(L(M_1)) \subseteq L(M_2)$.

Assume that (σ, τ) is an isomorphism, and consider a word $v_1 v_2 \dots v_m \in L(M_2)$. We will show that there is a unique word $w_1 w_2 \dots w_{m'} \in L(M_1)$ such that $\tau(w_1 w_2 \dots w_{m'}) = v_1 v_2 \dots v_m$. Since $v_1 v_2 \dots v_m \in L(M_2)$, we have states $s_{2,0}, s_{2,1}, \dots, s_{2,m} \in S_2$, 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 (σ, τ) is an isomorphism, σ 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, τ 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 $w_1 w_2 \dots w_3$, 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 $w_1 w_2 \dots w_3 \in L(M_1)$, and $\tau(w_1 w_2 \dots w_m) = v_1 v_2 \dots v_m$.

Now assume that there is another word $w'_1 w'_2 \dots w'_{m'} \in L(M_1)$ such that $\tau(w'_1 w'_2 \dots w'_{m'}) = v_1 v_2 \dots v_m$. Then we have states $s'_{1,0}, s'_{1,1}, \dots, s'_{1,m'} \in S_1$ 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 $w'_i = w_i$, so $w'_1 w'_2 \dots w'_{m'} = w_1 w_2 \dots w_m$. □

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

$$\tau(w) = \begin{cases} w & w \notin g(S_0), \\ \textit{undefined} & \textit{otherwise}, \end{cases}$$

is a bijection.

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

We construct the string $v^{(i)}$ from $v^{(i-1)}$. Suppose $\tau(v^{(i-1)}) = w_1 w_2 \dots w_{i-1}$, and that both M and M' are in state s after processing $v^{(i-1)}$ and $w_1 \dots w_{i-1}$, 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

$$\tau(v^{(i)}) = \tau(v^{(i-1)})\tau(w_i) = w_1 \dots w_i,$$

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

$$\tau(v^{(i)}) = \tau(v^{(i-1)})\tau(w_i)\tau(w'_i) = w_1 \dots w_i,$$

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 $v^{(0)}, v^{(1)}, \dots, v^{(n)}$ 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 τ is onto. Also, v is because of the functions g and h the only possible preimage of w, hence τ is also injective.

It remains to prove that $\tau(L(M)) \subseteq L(M')$. Assume that $v_1 v_2 \dots v_m \in L(M)$. Suppose $\tau(v_1 v_2 \dots v_{i-1}) = w^{(i-1)}$, and that both M and M' are in state $s \notin S_0$ after processing $v_1 v_2 \dots v_{i-1}$ 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 $v_1 v_2 \dots v_i$ and $\tau(v_1 v_2 \dots v_i) = w^{(i-1)} v_i$, 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 $v_1 v_2 \dots v_{i+1}$ and $\tau(v_1 v_2 \dots v_{i+1}) = w^{(i-1)} v_i$, 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 $v_1 v_2 \dots v_m$. Since $v_1 v_2 \dots v_m \in L(M)$, 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 $v_1 v_2 \dots v_m$ and $\tau(v_1 v_2 \dots v_m)$, respectively. Since M

is in a wait state at this point, so is M', and we conclude that $\tau(v_1 v_2 \dots v_m) \in L(M')$. □

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 τ 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 τ 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

$$\begin{aligned} f_{12} = \{((s_{1i}, s_{2k}), w, (s_{1j}, s_{2l})) \mid \\ &((s_{1i}, w, s_{1j}) \in f_1 \vee (w \notin I_1 \wedge s_{1i} = s_{1j} \wedge s_{1i} \in W_1)) \wedge \\ &((s_{2k}, w, s_{2l}) \in f_2 \vee (w \notin I_2 \wedge s_{2k} = s_{2l} \wedge s_{2k} \in W_2))\}. \end{aligned}$$

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 σ 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 $minimal(M) = (\hat{S}, \hat{I}, \hat{f}, \hat{W}, s_0)$, 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 $\tilde{M} = (\tilde{S}, \tilde{I}, \tilde{f}, \tilde{W}, s_0)$ is said to *include minimal*(M) if $\hat{S} \subseteq \tilde{S} \subseteq S$, $\hat{I} \subseteq \tilde{I} \subseteq I$, $\tilde{f}$ contains exactly those triples (s, w, s') in f such that $s \in \tilde{S}$ and $w \in \tilde{I}$, and $\tilde{W}$ consists of the wait states contained in $\tilde{S}$. For ease of presentation, we will sometimes speak of the machine M *restricted* to some subset $\tilde{S}$ of S, implying that the necessary changes are made to f and W in order to satisfy the above conditions.

Clearly, any machine including *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 *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 minimal(M), and let I be a subset of the input alphabet of 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 minimal(M_1) and minimal(M_2), respectively, and let I be a common subset of the input alphabets of minimal(M_1) and 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 $(A, B, \mathbf{Msg.Name}, \mathit{content})$. 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 $\mathcal{Z}$'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 $\mathcal{Z}$. Under this assumption, if we let I be the set of messages where $\mathcal{Z}$ is either the sender or recipient, then the restricted language $L|_I$ completely describes $\mathcal{Z}$'s information.

Suppose we have two protocols π_1 and π_2, and $\mathcal{Z}$ is trying to determine which protocol it is talking to. This gives rise to two systems, one with $\mathcal{Z}$ and the Turing machines of π_1, and one with $\mathcal{Z}$ and the Turing machines of π_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 $\mathcal{Z}$ in these systems.

Note that if a protocol's Turing machines are all deterministic, the protocol's responses to $\mathcal{Z}$'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 $\mathcal{Z}$, the next response by the protocols must be the same. This means that no matter what input $\mathcal{Z}$ gives to the protocols, the responses will always be identical. In other words, it will be impossible for $\mathcal{Z}$ to decide which protocol it is talking to. We have therefore proved the following theorem:

Theorem 9. *Let π_1 and π_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 $\mathcal{Z}$ interacts with π_1 and π_2, respectively. Let I_1 and I_2 be the sets of possible messages to and from $\mathcal{Z}$ in these systems. If $I_1 = I_2 = I$ and $L_1|_I = L_2|_I$, then π_1 securely realizes π_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 π_1 realizes π_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 π_{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

s	conditions/w	s'
(s_0, L)	$(B, A, \mathbf{B.Add}, m)$	$(s_1, L \cup \{m\})$
(s_0, L)	$\frac{m \in L, m \sim p}{(B, A, \mathbf{B.Lookup}, p, m)}$	(s_0, L)
(s_0, L)	$\frac{\forall m \in L : m \not\sim p}{(B, A, \mathbf{B.Lookup}, p, \perp)}$	(s_0, L)

FIGURE 1. Buffer machine with identifier A.

that recognize the possible communication patterns arising from an environment $\mathcal{Z}$ interacting either with the hybrid protocol π_{SM} or the ideal functionality $\mathcal{F}_{SM}$.

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

$$(s, w, s') \text{ and } (s, \text{conditions}/w, s'),$$

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 Σ 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 buffer machine M_A is $(\{s_0\} \times 2^T, I, f, \{s_0\} \times 2^T, (s_0, \emptyset))$, where the state transition function f is described in Figure 1. The input alphabet I is implicit in the description of f. (The buffer machines will always be "internal" to the Turing machines. Therefore we do not need to code messages as query-response pairs, we can encode the query and the response in the same message.)

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

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

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

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

FIGURE 2. The PKI functionality $\mathcal{F}_{PKI}$.

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

s	w	s'
s_0	$(P_i, \mathcal{F}_{PKI}, \textbf{PKI.Register}, v)$	(s_1, v)
s_1	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \textbf{PKI.Register}, P_i, v)$	(s_2, v)
s_2	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \textbf{PKI.Register.Ok}, P_i)$	(s_3, v)
s_3	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B.Add}, (P_i, v))$	$(s_4, -)$
s_4	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI.Register.Ok})$	$(s_5, -)$
s_5	$(P_i, \mathcal{F}_{PKI}, \textbf{PKI.Retrieve}, P_j)$	(s_6, P_j)
s_6	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \textbf{PKI.Retrieve}, P_i, P_j)$	(s_7, P_j)
s_7	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \textbf{PKI.Retrieve.Ok}, P_i, P_j)$	(s_8, P_j)
s_8	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B.Lookup}, (P_j, -), (P_j, v))$	(s_9, v)
s_9	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI.Retrieve.Ok}, v)$	$(s_5, -)$
s_8	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B.Lookup}, (P_j, -), \perp)$	$(s_{10}, -)$
s_{10}	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI.Retrieve.Fail})$	$(s_5, -)$

FIGURE 3. PKI per-player machine for player i.

machine $M_{SC}(i)$ is

$$(\{s_0, s_1, \dots, s_{19}\} \times (T \cup \{-\}), I, f, \{s_0, s_2, s_5, s_8, s_{10}, s_{14}\} \times (T \cup \{-\}), (s_0, -)),$$

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 π_{SM} are given in Figure 6. The machine $M_\pi(i)$ modeling the protocol machine used by the i'th player is

$$(\{s_0, s_1, \dots, s_{20}\} \times (T \cup \{-\}), I, f, \{s_0, s_2, s_4, s_6, s_8, s_{10}, s_{15}, s_{18}\} \times (T \cup \{-\}), (s_0, -)),$$

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

$$(\{s_0, s_1, s_2\} \times (2^T \cup \{-\}), I, f, \{(s_0, -)\}, (s_0, -)),$$

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

$$(\{s_0, s_1, \dots, s_{19}\} \times (T \cup \{-\}), I, f, \{s_0, s_2, s_5, s_7, s_{13}\} \times (T \cup \{-\}), (s_0, -)),$$

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 $B_{\mathcal{S}}$. The per-player machine $M_{\mathcal{S}}(i)$ is

$$(\{s_0, s_1, \dots, s_{19}\} \times (T \cup \{-\}), I, f, \{s_0, s_2, \} \times (T \cup \{-\}), (s_0, -)),$$

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

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

When **SC**.**KeyGen** is received from $\tilde{P}_i$:

1. If $\mathcal{F}_{SC}$ has already processed an **SC**.**KeyGen** message from $\tilde{P}_i$, send **SC**.**Error** to $\tilde{P}_i$ and stop.
2. Send $(\mathbf{SC}.\mathbf{KeyGen}, P_i)$ to the adversary $\mathcal{A}$ and wait for $(\mathbf{SC}.\mathbf{Key}, P_i, pk_i^s, pk_i^r)$ from $\mathcal{A}$. (We note that $\mathcal{A}$ 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 $\mathcal{A}$ and $\mathcal{F}_{SC}$.)
3. Store (P_i, pk_i^s, pk_i^r) in the public key buffer and send $(\mathbf{SC}.\mathbf{Key}, pk_i^s, pk_i^r)$ to $\tilde{P}_i$.

When $(\mathbf{SC}.\mathbf{Encrypt}, pk^r, m)$ 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 **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 $(\mathbf{SC}.\mathbf{Encrypt}, pk_i^s, pk^r, |m|)$ to $\mathcal{A}$ and wait for $(\mathbf{SC}.\mathbf{Ciphertext}, pk_i^s, pk^r, c)$ 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 **SC**.**Ciphertext** message from $\mathcal{A}$ to $\mathcal{F}_{SC}$, nor in an **SC**.**Decrypt** message to $\mathcal{A}$ from $\mathcal{F}_{SC}$.)
   (b) Store (pk_i^s, pk^r, c, m) in the ciphertext buffer, send $(\mathbf{SC}.\mathbf{Ciphertext}, c)$ to $\tilde{P}_i$ and stop.
3. Send $(\mathbf{SC}.\mathbf{Encrypt}, pk_i^s, pk^r, m)$ to $\mathcal{A}$ and wait for $(\mathbf{SC}.\mathbf{Ciphertext}, pk_i^s, pk^r, c)$ from $\mathcal{A}$. (We note that the above restrictions on c also apply here.)
4. Send $(\mathbf{SC}.\mathbf{Ciphertext}, c)$ to $\tilde{P}_i$.

When $(\mathbf{SC}.\mathbf{Decrypt}, pk^s, c)$ 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 **SC**.**Error** to $\tilde{P}_j$ and stop.
2. Send $(\mathbf{SC}.\mathbf{Decrypt}, pk^s, pk_j^r, c)$ to $\mathcal{A}$ and wait for $(\mathbf{SC}.\mathbf{Plaintext}, pk^s, pk_j^r, m')$ from $\mathcal{A}$.
3. If there is an entry (pk^s, pk_j^r, c, m) in the ciphertext buffer, send $(\mathbf{SC}.\mathbf{Plaintext}, m)$ to $\tilde{P}_j$ and stop.
4. If there is an entry $(P_i, pk^s, \cdot)$ in the public key buffer, send $(\mathbf{SC}.\mathbf{Plaintext}, \perp)$ to $\tilde{P}_j$ and stop.
5. Send $(\mathbf{SC}.\mathbf{Plaintext}, m')$ to $\tilde{P}_j$.

FIGURE 4. The signcryption functionality $\mathcal{F}_{SC}$.

Composite Machines. We can now compose these building blocks into two machines,

$$M_1' = M_{SC} + M_{PKI} + \sum_i M_\pi(i) \text{ and } M_2' = M_{SM} + M_{\mathcal{S}}.$$

s	w	s'
s_0	$(P_i, \mathcal{F}_{SC}, \textbf{SC}.\textbf{KeyGen})$	$(s_1, -)$
s_1	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC}.\textbf{KeyGen}, P_i)$	$(s_2, -)$
s_2	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC}.\textbf{Key}, P_i, pk_i^s, pk_i^r)$	(s_3, pk_i^s, pk_i^r)
s_3	$(\mathcal{F}_{SC}, B_{SC}, \textbf{B}.\textbf{Add}, (P_i, pk_i^s, pk_i^r))$	(s_4, pk_i^s, pk_i^r)
s_4	$(\mathcal{F}_{SC}, P_i, \textbf{SC}.\textbf{Key}, pk_i^s, pk_i^r)$	(s_5, pk_i^s, pk_i^r)
s_5	$(P_i, \mathcal{F}_{SC}, \textbf{SC}.\textbf{Encrypt}, pk^r, m)$	$(s_6, pk_i^s, pk_i^r, pk^r, m)$
s_6	$(\mathcal{F}_{SC}, B_{SC}, \textbf{B}.\textbf{Lookup}, (-, -, pk^r), \bot)$	$(s_7, pk_i^s, pk_i^r, pk^r, m)$
s_6	$(\mathcal{F}_{SC}, B_{SC}, \textbf{B}.\textbf{Lookup}, (-, -, pk^r), (P_j, pk_j^s, pk^r))$	$(s_9, pk_i^s, pk_i^r, pk^r, m)$
s_7	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC}.\textbf{Encrypt}, pk_i^s, pk^r, m)$	$(s_8, pk_i^s, pk_i^r, pk^r)$
s_8	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC}.\textbf{Ciphertext}, pk_i^s, pk^r, c)$	$(s_{12}, pk_i^s, pk_i^r, c)$
s_9	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC}.\textbf{Encrypt}, pk_i^s, pk^r, \|m\|)$	$(s_{10}, pk_i^s, pk_i^r, pk^r, m)$
s_{10}	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC}.\textbf{Ciphertext}, pk_i^s, pk^r, c)$	$(s_{11}, pk_i^s, pk_i^r, pk^r, m, c)$
s_{11}	$(\mathcal{F}_{SC}, B_{SC}, \textbf{B}.\textbf{Add}, (pk_i^s, pk^r, c, m))$	$(s_{12}, pk_i^s, pk_i^r, c)$
s_{12}	$(\mathcal{F}_{SC}, P_i, \textbf{SC}.\textbf{Ciphertext}, c)$	(s_5, pk_i^s, pk_i^r)
s_5	$(P_i, \mathcal{F}_{SC}, \textbf{SC}.\textbf{Decrypt}, pk^s, c)$	$(s_{13}, pk_i^s, pk_i^r, pk^s, c)$
s_{13}	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC}.\textbf{Decrypt}, pk^s, pk_i^r, c)$	$(s_{14}, pk_i^s, pk_i^r, pk^s, c)$
s_{14}	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC}.\textbf{Plaintext}, pk^s, pk_i^r, m')$	$(s_{15}, pk_i^s, pk_i^r, pk^s, c, m')$
s_{15}	$(\mathcal{F}_{SC}, B_{SC}, \textbf{B}.\textbf{Lookup}, (-, pk^s, -), \bot)$	$(s_{16}, pk_i^s, pk_i^r, m')$
s_{15}	$(\mathcal{F}_{SC}, B_{SC}, \textbf{B}.\textbf{Lookup}, (-, pk^s, -), (P_j, pk^s, pk_j^r))$	$(s_{17}, pk_i^s, pk_i^r, pk_j^s, c)$
s_{16}	$(\mathcal{F}_{SC}, P_i, \textbf{SC}.\textbf{Plaintext}, m')$	(s_5, pk_i^s, pk_i^r)
s_{17}	$(\mathcal{F}_{SC}, B_{SC}, \textbf{B}.\textbf{Lookup}, (pk^s, pk_i^r, c, -), \bot)$	$(s_{18}, pk_i^s, pk_i^r . \bot)$
s_{17}	$(\mathcal{F}_{SC}, B_{SC}, \textbf{B}.\textbf{Lookup}, (pk^s, pk_i^r, c, -), (pk^s, pk_i^r, c, m))$	$(s_{19}, pk_i^s, pk_i^r, m)$
s_{18}	$(\mathcal{F}_{SC}, P_i, \textbf{SC}.\textbf{Plaintext}, \bot)$	(s_5, pk_i^s, pk_i^r)
s_{19}	$(\mathcal{F}_{SC}, P_i, \textbf{SC}.\textbf{Plaintext}, m)$	(s_5, pk_i^s, pk_i^r)

FIGURE 5. Signcryption per-player machine for player i.

4. EQUIVALENCE PROOF

This section is dedicated to proving our main result.

Theorem 10. *The protocol π_{SM} securely realizes $\mathcal{F}_{SM}$ in the $(\mathcal{F}_{PKI}, \mathcal{F}_{SC})$-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

$$M_2 = B_{SM} + B_{\mathcal{S}} + \sum_i (M_{SM}(i) + M_{\mathcal{S}}(i)).$$

When P_i receives **SM.Register** from the environment:

1. If P_i has already processed **SM.Register**, send **SM.Error** to the environment and stop.
2. Send **SC.KeyGen** to $\mathcal{F}_{SC}$ and wait for $(\mathbf{SC.Key}, pk_i^s, pk_i^r)$ from $\mathcal{F}_{SC}$.
3. Send $(\mathbf{PKI.Register}, (pk_i^s, pk_i^r))$ to $\mathcal{F}_{PKI}$ and wait for **PKI.Registered** from $\mathcal{F}_{PKI}$.
4. Send **SM.Register.Ok** to the environment.

When P_i receives $(\mathbf{SM.Encrypt}, m)$ from the environment:

1. If P_i has not processed **SM.Register**, send **SM.Error** to the environment and stop.
2. Send $(\mathbf{PKI.Retrieve}, P_j)$ to $\mathcal{F}_{PKI}$ and wait.
3. If $\mathcal{F}_{PKI}$ replies with $(\mathbf{PKI.Retrieved}, \perp)$, send **SM.Not.Registered** to the environment and stop.
4. Otherwise $\mathcal{F}_{PKI}$ replies with $(\mathbf{PKI.Retrieved}, (\cdot, pk_j^r))$.
5. Send $(\mathbf{SC.Encrypt}, pk_j^r, m)$ to $\mathcal{F}_{SC}$ and wait for $(\mathbf{SC.Ciphertext}, c)$ from $\mathcal{F}_{SC}$.
6. Send $(\mathbf{SM.Ciphertext}, c)$ to the environment.

When P_j receives $(\mathbf{SM.Decrypt}, c)$ from the environment:

1. If P_j has not processed **SM.Register**, send **SM.Error** to the environment and stop.
2. Send $(\mathbf{PKI.Retrieve}, P_i)$ to $\mathcal{F}_{PKI}$.
3. If $\mathcal{F}_{PKI}$ replies with $(\mathbf{PKI.Retrieved}, \perp)$, send **SM.Not.Registered** to the environment and stop.
4. Otherwise $\mathcal{F}_{PKI}$ replies with $(\mathbf{PKI.Retrieved}, (pk_i^s, \cdot))$.
5. Send $(\mathbf{SC.Decrypt}, pk_i^s, c)$ to $\mathcal{F}_{SC}$ and wait for $(\mathbf{SC.Plaintext}, m)$ from $\mathcal{F}_{SC}$.
6. Send $(\mathbf{SM.Plaintext}, m)$ to the environment.

FIGURE 6. The secure messaging protocol π_{SM}.

By Lemma 5, M_1' and M_1 are isomorphic. In particular, it is easy to define an isomorphism $(\sigma, \tau) : M_1' \to M_1$, where τ 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 $M_{SC}(i) + M_\pi(i) + M_{PKI}(i)$ when running

s	w	s'
s_0	$(\mathcal{Z}, P_i, \mathbf{SM.Register})$	$(s_1, -)$
s_1	$(P_i, \mathcal{F}_{SC}, \mathbf{SC.KeyGen})$	$(s_2, -)$
s_2	$(\mathcal{F}_{SC}, P_i, \mathbf{SC.Key}, pk^s, pk^r)$	(s_3, pk^s, pk^r)
s_3	$(P_i, \mathcal{F}_{PKI}, \mathbf{PKI.Register}, (pk^s, pk^r))$	$(s_4, -)$
s_4	$(\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Register.Ok})$	$(s_5, -)$
s_5	$(P_i, \mathcal{Z}, \mathbf{SM.Register.Ok})$	$(s_6, -)$
s_6	$(\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$	(s_7, P_j, m)
s_7	$(P_i, \mathcal{F}_{PKI}, \mathbf{PKI.Retrieve}, P_j)$	(s_8, m)
s_8	$(\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Retrieve.Fail})$	$(s_9, -)$
s_8	$(\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Retrieve.Ok}, (pk_j^s, pk_j^r))$	(s_{10}, pk_j^r, m)
s_9	$(P_i, \mathcal{Z}, \mathbf{SM.Not.Registered})$	$(s_6, -)$
s_{10}	$(P_i, \mathcal{F}_{SC}, \mathbf{SC.Encrypt}, pk_j^r, m)$	$(s_{11}, -)$
s_{11}	$(\mathcal{F}_{SC}, P_i, \mathbf{SC.Ciphertext}, c)$	(s_{12}, c)
s_{12}	$(P_i, \mathcal{Z}, \mathbf{SM.Ciphertext}, c)$	$(s_6, -)$
s_6	$(\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$	(s_{13}, P_j, c)
s_{13}	$(P_i, \mathcal{F}_{PKI}, \mathbf{PKI.Retrieve}, P_j)$	(s_{14}, c)
s_{14}	$(\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Retrieve.Fail})$	$(s_{15}, -)$
s_{14}	$(\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Retrieve.Ok}, (pk_j^s, pk_j^r))$	(s_{16}, pk_j^s, c)
s_{15}	$(P_i, \mathcal{Z}, \mathbf{SM.Not.Registered})$	$(s_6, -)$
s_{16}	$(P_i, \mathcal{F}_{SC}, \mathbf{SC.Decrypt}, pk_j^s, c)$	$(s_{17}, -)$
s_{17}	$(\mathcal{F}_{SC}, P_i, \mathbf{SC.Plaintext}, m)$	(s_{18}, m)
s_{18}	$(P_i, \mathcal{Z}, \mathbf{SM.Plaintext}, m)$	$(s_6, -)$
s_0	$(\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$	$(s_{19}, -)$
s_0	$(\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$	$(s_{19}, -)$
s_6	$(\mathcal{Z}, P_i, \mathbf{SM.Register})$	$(s_{20}, -)$
s_{19}	$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(s_0, -)$
s_{20}	$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(s_6, -)$

FIGURE 7. Protocol machine for player i.

s	w	s'
s_0	$(\mathcal{Z}, P_i, x)$	(s_1, x)
s_1	$(P_i, \mathcal{F}_{SM}, x)$	$(s_0, -)$
s_0	$(\mathcal{F}_{SM}, P_i, x)$	(s_2, x)
s_2	$(P_i, \mathcal{Z}, x)$	$(s_0, -)$

FIGURE 8. Dummy machine for player i.

When **SM.Register** is received from $\tilde{P}_i$:

(1) If $\mathcal{F}_{SM}$ has already processed an **SM.Register** message from $\tilde{P}_i$, send **SM.Error** to $\tilde{P}_i$ and stop.
(2) Send (**SM.Register**, P_i) to the adversary $\mathcal{A}$ and wait for (**SM.Register.OK**, P_i) from $\mathcal{A}$.
(3) Store P_i in the registration buffer and send **SM.Register.Ok** to $\tilde{P}_i$.

When (**SM.Encrypt**, P_j, m) is received from $\tilde{P}_i$:

(1) If P_i is not stored in the registration buffer, send **SM.Error** to $\tilde{P}_i$ and stop.
(2) Send (**SM.Encrypt**, $P_i, P_j, |m|$) to the adversary $\mathcal{A}$ and wait for (**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 **SM.Ciphertext** message from $\mathcal{A}$ to $\mathcal{F}_{SM}$, nor in an **SM.Decrypt** message to $\mathcal{A}$ from $\mathcal{F}_{SM}$ that resulted in an **SM.Plaintext** being sent from $\mathcal{F}_{SM}$.)
(3) If P_j is not stored in the registration buffer, send **SM.Not.Registered** to $\tilde{P}_i$ and stop.
(4) Store (P_i, P_j, c, m) in the ciphertext buffer and send (**SM.Ciphertext**, c) to $\tilde{P}_i$.

When (**SM.Decrypt**, P_i, c) is received from $\tilde{P}_j$:

(1) If P_j is not stored in the registration buffer, send **SM.Error** to $\tilde{P}_j$ and stop.
(2) Send (**SM.Decrypt**, P_j, P_i, c) to the adversary $\mathcal{A}$ and wait for (**SM.Plaintext**, P_j, P_i, m') from $\mathcal{A}$.
(3) If P_i is not stored in the registration buffer, send **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 $(SM.Plaintext, m)$ to $\tilde{P}_j$, otherwise send $(SM.Plaintext, \perp)$ 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 $M_{SM}(i) + M_{\mathcal{S}}(i)$ 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 $M_{SC}(i)+ M_{\pi}(i) + M_{PKI}(i)$ and $M_{SM}(i) + M_{\mathcal{S}}(i)$ given in the figures are evident from the descriptions of the individual machines. However, we briefly comment on the

s	w	s'
s_0	$(\mathcal{Z}, P_i, \mathbf{SM.Register})$	$(s_1, -)$
s_1	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Register}, P_i)$	$(s_2, -)$
s_2	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Register.Ok}, P_i)$	$(s_3, -)$
s_3	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Add}, P_i)$	$(s_4, -)$
s_4	$(P_i, \mathcal{Z}, \mathbf{SM.Register.Ok})$	$(s_5, -)$
s_5	$(\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$	(s_6, P_j, m)
s_6	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Encrypt}, P_i, P_j, \|m\|)$	(s_7, P_j, m)
s_7	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Ciphertext}, P_i, P_j, c)$	(s_8, P_j, m, c)
s_8	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, \perp)$	$(s_9, -)$
s_8	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, P_j)$	(s_{10}, P_j, m, c)
s_9	$(P_i, \mathcal{Z}, \mathbf{SM.Not.Registered})$	$(s_5, -)$
s_{10}	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Add}, (P_i, P_j, c, m))$	(s_{11}, c)
s_{11}	$(P_i, \mathcal{Z}, \mathbf{SM.Ciphertext}, c)$	$(s_5, -)$
s_5	$(\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$	(s_{12}, P_j, c)
s_{12}	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Decrypt}, P_i, P_j, c)$	(s_{13}, P_j, c)
s_{13}	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Plaintext}, P_i, P_j, m')$	(s_{14}, P_j, c, m')
s_{14}	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, \perp)$	$(s_{15}, -, m')$
s_{14}	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, P_j)$	(s_{16}, P_j, c)
s_{15}	$(P_i, \mathcal{Z}, \mathbf{SM.Not.Registered})$	$(s_5, -)$
s_{16}	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, (P_i, P_j, c, -), \perp)$	$(s_{17}, \perp)$
s_{16}	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, (P_i, P_j, c, -), (P_i, P_j, c, m))$	(s_{18}, m)
s_{17}	$(P_i, \mathcal{Z}, \mathbf{SM.Plaintext}, \perp)$	$(s_5, -)$
s_{18}	$(P_i, \mathcal{Z}, \mathbf{SM.Plaintext}, m)$	$(s_5, -)$
s_0	$(\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$	$(s_{19}, -)$
s_0	$(\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$	$(s_{19}, -)$
s_5	$(\mathcal{Z}, P_i, \mathbf{SM.Register})$	$(s_{20}, -)$
s_{19}	$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(s_0, -)$
s_{20}	$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(s_5, -)$

FIGURE 10. Secure messaging per-player machine for player i.

messages containing **B.Lookup**, since transitions involving such messages depend on the state of some buffer machine. We start by considering the message $(\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B.Lookup}, (P_j, -), \perp/(P_j, (pk^s_j, pk^r_j)))$, 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 $(P_j, (pk^s_j, pk^r_j))$ 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 $(\mathcal{F}_{PKI}, B_{PKI}$, $\mathbf{B.Lookup}, (P_j, -), (P_j, v))$ is recognized only if $v = (pk^s_j, pk^r_j)$, hence we need only include $(\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B.Lookup}, (P_j, -), \perp/(P_j, (pk^s_j, pk^r_j)))$ in the figures. For ease of presentation, Figures 14 and 15 only include the case where P_j

When $\mathcal{S}$ receives $(\mathbf{SM.Register}, P_i)$ from $\mathcal{F}_{SM}$:

1. Send $(\mathbf{SC.KeyGen}, P_i)$ to $\mathcal{A}_{SC}$ and wait for $(\mathbf{SC.Key}, P_i, pk_i^s, pk_i^r)$. (We note that $\mathcal{A}_{SC}$ 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 $\mathcal{A}_{SC}$ and $\mathcal{S}$.)
2. Send $(\mathbf{PKI.Register}, (P_i, pk_i^s, pk_i^r))$ to $\mathcal{A}_{PKI}$ and wait for $(\mathbf{PKI.Register.Ok}, P_i)$ from $\mathcal{A}_{PKI}$.
3. Store (P_i, pk_i^s, pk_i^r) in the public key buffer.
4. Send $(\mathbf{SM.Register.Ok}, P_i)$ to $\mathcal{F}_{SM}$.

When $\mathcal{S}$ receives $(\mathbf{SM.Encrypt}, P_i, P_j, |m|)$ from $\mathcal{F}_{SM}$:

1. Send $(\mathbf{PKI.Retrieve}, P_i, P_j)$ to $\mathcal{A}_{PKI}$ and wait for $(\mathbf{PKI.Retrieve.Ok}, P_i, P_j)$ from $\mathcal{A}_{PKI}$.
2. If there is no entry $(P_j, \cdot, \cdot)$ in public key buffer, send $(\mathbf{SM.Ciphertext}, P_i, P_j, \perp)$ to $\mathcal{F}_{SM}$ and stop.
3. Find entries $(P_i, pk_i^s, \cdot)$ and $(P_j, \cdot, pk_j^r)$ in public key buffer, send $(\mathbf{SC.Encrypt}, pk_i^s, pk_j^r, |m|)$ to $\mathcal{A}_{SC}$, and wait for $(\mathbf{SC.Ciphertext}, pk_i^s, pk_j^r, c)$ from $\mathcal{A}_{SC}$.
4. Send $(\mathbf{SM.Ciphertext}, P_i, P_j, c)$ to $\mathcal{F}_{SM}$.

When $\mathcal{S}$ receives $(\mathbf{SM.Decrypt}, P_j, P_i, c)$ from $\mathcal{F}_{SM}$:

1. Send $(\mathbf{PKI.Retrieve}, P_j, P_i)$ to $\mathcal{A}_{PKI}$ and wait for $(\mathbf{PKI.Retrieve.Ok}, P_j, P_i)$ from $\mathcal{A}_{PKI}$.
2. If there is no entry $(P_i, \cdot, \cdot)$ in public key buffer, send $(\mathbf{SM.Plaintext}, P_j, P_i, \perp)$ to $\mathcal{F}_{SM}$ and stop.
3. Find entries $(P_i, pk_i^s, \cdot)$ and $(P_j, \cdot, pk_j^r)$ in public key buffer, send $(\mathbf{SC.Decrypt}, pk_i^s, pk_j^r, c)$ to $\mathcal{A}_{SC}$, and wait for $(\mathbf{SC.Plaintext}, pk_i^s, pk_j^r, m')$ from $\mathcal{A}_{SC}$.
4. Send $(\mathbf{SM.Plaintext}, P_j, P_i, m')$ to $\mathcal{F}_{SM}$.

FIGURE 11. The ideal adversary $\mathcal{S}$.

has registered, while the case where P_j has not registered is handled in Figure 16. The same reasoning applies to the message $(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Lookup}, (P_j, -), \perp/(P_j, (pk_j^s, pk_j^r)))$, which is treated in the same way.

We proceed by considering the message $(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (-, -, pk_j^r), (P_j, pk_j^s, pk_j^r))$, which can be recognized by $M_{SC}(i)$ in state $(s_6, pk_i^s, pk_i^r, pk_j^r, m)$. By tracing backwards, we see that for $M_{SC}(i)$ to reach this state, there must be a pair $(P_j, (pk_j^s, pk_j^r))$ 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^r). Since public keys are assumed to be unique, this is the only triple containing pk_j^r in the state of B_{SC}. This means that $(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (-, -, pk_j^r), \perp)$ is not recognized, and that $(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (-, -, pk_j^r), (P_k, pk_k^s, pk_j^r))$ is recognized if and only if $P_k = P_j$ and $pk_k^s = pk_j^s$, hence we need only include $(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup},$

s	w	s'
s_0	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Register}, P_i)$	$(s_1, -)$
s_1	$(\mathcal{S}, \mathcal{A}_{SC}, \mathbf{SC.KeyGen}, P_i)$	$(s_2, -)$
s_2	$(\mathcal{A}_{SC}, \mathcal{S}, \mathbf{SC.Key}, P_i, pk_i^s, pk_i^r)$	(s_3, pk_i^s, pk_i^r)
s_3	$(\mathcal{S}, \mathcal{A}_{PKI}, \mathbf{PKI.Register}, P_i, pk_i^s, pk_i^r)$	(s_4, pk_i^s, pk_i^r)
s_4	$(\mathcal{A}_{PKI}, \mathcal{S}, \mathbf{PKI.Register.Ok}, P_i)$	(s_5, pk_i^s, pk_i^r)
s_5	$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Add}, (P_i, (pk_i^s, pk_i^r)))$	(s_6, pk_i^s, pk_i^r)
s_6	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Register.Ok}, P_i)$	(s_7, pk_i^s, pk_i^r)
s_7	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Encrypt}, P_i, P_j, \vert m\vert)$	$(s_8, pk_i^s, pk_i^r, P_j, \vert m\vert)$
s_8	$(\mathcal{S}, \mathcal{A}_{PKI}, \mathbf{PKI.Retrieve}, P_i, P_j)$	$(s_9, pk_i^s, pk_i^r, P_j, \vert m\vert)$
s_9	$(\mathcal{A}_{PKI}, \mathcal{S}, \mathbf{PKI.Retrieve.Ok}, P_i, P_j)$	$(s_{10}, pk_i^s, pk_i^r, P_j, \vert m\vert)$
s_{10}	$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Lookup}, (P_j, -), \perp)$	$(s_{11}, pk_i^s, pk_i^r, P_j)$
s_{10}	$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$	$(s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, \vert m\vert)$
s_{11}	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Ciphertext}, P_i, P_j, \perp)$	(s_7, pk_i^s, pk_i^r)
s_{12}	$(\mathcal{S}, \mathcal{A}_{SC}, \mathbf{SC.Encrypt}, pk_i^s, pk_j^r, \vert m\vert)$	$(s_{13}, pk_i^s, pk_i^r, P_j, pk_j^r)$
s_{13}	$(\mathcal{A}_{SC}, \mathcal{S}, \mathbf{SC.Ciphertext}, pk_i^s, pk_j^r, c)$	$(s_{14}, pk_i^s, pk_i^r, P_j, c)$
s_{14}	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Ciphertext}, P_i, P_j, c)$	(s_7, pk_i^s, pk_i^r)
s_7	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Decrypt}, P_i, P_j, c)$	$(s_{15}, pk_i^s, pk_i^r, P_j, c)$
s_{15}	$(\mathcal{S}, \mathcal{A}_{PKI}, \mathbf{PKI.Retrieve}, P_i, P_j)$	$(s_{16}, pk_i^s, pk_i^r, P_j, c)$
s_{16}	$(\mathcal{A}_{PKI}, \mathcal{S}, \mathbf{PKI.Retrieve.Ok}, P_i, P_j)$	$(s_{17}, pk_i^s, pk_i^r, P_j, c)$
s_{17}	$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Lookup}, (P_j, -), \perp)$	$(s_{18}, pk_i^s, pk_i^r, P_j)$
s_{17}	$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$	$(s_{19}, pk_i^s, pk_i^r, P_j, pk_j^s, c)$
s_{18}	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Plaintext}, P_i, P_j, \perp)$	(s_7, pk_i^s, pk_i^r)
s_{19}	$(\mathcal{S}, \mathcal{A}_{SC}, \mathbf{SC.Decrypt}, pk_j^s, pk_i^r, c)$	$(s_{20}, pk_i^s, pk_i^r, P_j, pk_j^s)$
s_{20}	$(\mathcal{A}_{SC}, \mathcal{S}, \mathbf{SC.Plaintext}, pk_j^s, pk_i^r, m')$	$(s_{21}, pk_i^s, pk_i^r, P_j, m')$
s_{21}	$(\mathcal{S}, \mathbf{SM.Plaintext}, P_i, P_j, m')$	(s_7, pk_i^s, pk_i^r)

FIGURE 12. Ideal adversary per-player machine for player i.

$(-, -, pk_j^r), (P_j, pk_j^s, pk_j^r))$ in Figure 14. Similar arguments apply to the messages $(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (-, pk_j^s, -), \perp/(P_l, pk_j^s, pk_l^r))$ and $(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, \perp/P_j)$.

Alternatively, we may view the description of $M_{SC}(i)+M_{PKI}(i)+M_\pi(i)$ when running in M_1 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_1. 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_1.

We now modify M_1 into a new machine $M_1^{(1)}$ by replacing $M_{SC}(i) + M_\pi(i) + M_{PKI}(i)$ in the above expression by $C_1(i)$, that is,

$$M_1^{(1)} = B_{SC} + B_{PKI} + \sum_i C_1(i).$$

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 *minimal*(M_1). Lemma 7 then implies that M_1 and $M_1^{(1)}$ are equivalent with respect to I. □

By essentially repeating the above procedure, M_2 is modified into a new machine $M_2^{(1)}$ by replacing $M_{SM}(i) + M_{\mathcal{S}}(i)$ by a machine $C_2(i)$, whose set of states and input alphabet contain the reachable states and the significant messages of $M_{SM}(i) + M_{\mathcal{S}}(i)$ when running in M_2. That is,

$$M_2^{(2)} = B_{SM} + B_{\mathcal{S}} + \sum_i C_2(i).$$

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 $(P_k, (pk_k^s, pk_j^r))$ 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 $(P_l, (pk_j^s, pk_l^r))$ 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 $P_k = P_l = P_j$.

Let M and M' be the machines $M_1^{(2)}$ and $M_1^{(1)}$ restricted to S and S', respectively, and let $\sigma : S \to S'$ 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, σ 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 σ. It can be verified that $(\sigma, \tau) : M \to M'$, where $\tau : I_1^{(2)} \to I_1^{(1)}$ is the identity map, is an isomorphism. It follows from Lemma 1 that the induced language map $\tau : L(M) \to L(M')$ is a bijection. τ leaves I fixed, so by Lemma 3, M and M' are equivalent with respect to I. Since M includes $minimal(M_1^{(2)})$ and M' includes $minimal(M_1^{(1)})$, Corollary 8 implies that $M_1^{(1)}$ and $M_1^{(2)}$ are equivalent with respect to I. □

As for $M_2^{(1)}$, this machine is modified into a new machine $M_2^{(2)}$ by adding the key pk_j^r from the pair $(P_j, (pk_j^s, pk_j^r))$ in the state of $B_{\mathcal{S}}$ to the following states: s_{14} of $M_{\mathcal{S}}(i)$ and s_8 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 $B_{\mathcal{S}}$ to the following states: s_{21} of $M_{\mathcal{S}}(i)$ 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 $M_2^{(1)}$ and $M_2^{(2)}$ are equivalent with respect to I. In particular, when considering only reachable states of $M_2^{(1)}$, pk_j^r and pk_j^s in the above description are well-defined.*

Proof. Let S and S' be the sets of reachable states of $S_2^{(2)}$ and $S_2^{(1)}$, 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 $B_{\mathcal{S}}$. 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 $M_2^{(2)}$ and $M_2^{(1)}$ restricted to S and S', respectively, and let $\sigma : S \to S'$ 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, σ removes the added key pk_j^r or pk_j^s from this state, and applies the identity map to the states of all other machines. Since the states from which pk_j^r or pk_j^s 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 $B_{\mathcal{S}}$, hence no information is lost when applying σ. It can be verified that $(\sigma, \tau) : M \to M'$, where $\tau : I_2^{(2)} \to I_2^{(1)}$ is the identity map, is an isomorphism. It follows from Lemma 1 that the induced language map $\tau : L(M) \to L(M')$ is a bijection. τ leaves I fixed, so by Lemma 3, M and

M' are equivalent with respect to I. Since M includes $\mathit{minimal}(M_2^{(2)})$ and M' includes $\mathit{minimal}(M_2^{(1)})$, Corollary 8 implies that $M_2^{(1)}$ and $M_2^{(2)}$ are equivalent with respect to I. $\square$

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 $M_1^{(2)}$ and $M_1^{(3)}$ 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, $I_1^{(4)} = I_1^{(3)} \cup \{(P, B_{SM}^*, \mathbf{B}.\mathbf{Add}, m)\} \cup \{(P, B_{SM}^*, \mathbf{B}.\mathbf{Lookup}, p, m)\} \cup \{(P, B_{SM}^*, \mathbf{B}.\mathbf{Lookup}, p, \perp)\}$.

Lemma 15. *The machines $M_1^{(3)}$ and $M_1^{(4)}$ are equivalent with respect to I.*

Proof. Note that $\mathit{dom}(f_1^{(3)})_2 = \mathit{dom}(f_1^{(4)})_2$, since the additional messages in $I^{(4)}$ have no transitions related to them. Let $(\sigma, \tau) : M_1^{(4)} \to M_1^{(3)}$ be defined as follows: $\sigma : S_1^{(4)} \to S_1^{(3)}$ is the identity map, while $\tau : I_2^{(4)} \to I_2^{(3)}$ applies the identity map to all messages except the messages meant for B_{SM}^*, for which it is undefined. It is clear that (σ, τ) is an isomorphism. Then, by Lemma 1, the induced language map $\tau : L(M_1^{(4)}) \to L(M_1^{(3)})$ is a bijection. τ leaves I fixed, and it follows from Lemma 3 that $M_1^{(3)}$ and $M_1^{(4)}$ are equivalent with respect to I. $\square$

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 $(P, B_{SM}^*, \mathbf{B}.\mathbf{Add}, m)$. 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. $w \notin (\mathit{dom} f_1)_2$, 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 $S_1^{(4)}$ of states of $M_1^{(4)}$ is given by

$$S_1^{(4)} = S_{B_{SC}} \times S_{B_{PKI}} \times \prod_{i=1}^{n} C_1(i),$$

so the set $S^{(5)}$ of states of $M_1^{(5)} = M_1^{(4)} + B_{SM}^*$ is given by

$$S_1^{(5)} = S_{B_{SM}^*} \times S_{B_{SC}} \times S_{B_{PKI}} \times \prod_{i=1}^{n} C_1(i).$$

From the above argument, we deduce that a subset S' containing all reachable states of $M_1^{(5)}$ 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 $M_1^{(5)}$ restricted to S', and let $(\sigma, \tau) : M' \to M_1^{(4)}$ be defined as follows: $\sigma : S' \to S_1^{(4)}$ applies the identity map to each state except that of B_{SM}^*, and $\tau : I_1^{(5)} \to I_1^{(4)}$ is the identity map. Note that, since $f_1^{(5)}$ has no transitions involving B_{SM}^*, it has essentially the same transitions as $f_1^{(4)}$. It can be verified that (σ, τ) is an isomorphism. By Lemma 1, the induced language map $\tau : L(M') \to L(M_1^{(4)})$ is a bijection. τ leaves I fixed, and it follows from Lemma 3 that $M_1^{(4)}$ and M' are equivalent with respect to I. Since M' includes *minimal*$(M_1^{(5)})$, Corollary 8 implies that $M_1^{(4)}$ and $M_1^{(5)}$ are equivalent with respect to I. □

We would also like to add a buffer machine B_{SC}^* to $M_2^{(3)}$, as an equivalent to B_{SC} in $M_1^{(3)}$. Before doing this, we must modify the input alphabet $I_2^{(3)}$: We create $M_2^{(4)}$ from $M_2^{(3)}$ by adding messages recognized by B_{SC}^* to $I_2^{(3)}$, that is, $I_2^{(4)} = I_2^{(3)} \cup \{(P, B_{SC}^*, \mathbf{B}.\mathbf{Add}, m)\} \cup \{(P, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, p, m)\} \cup \{(P, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, p, \perp)\}$. The proof of the following lemma is similar to the proof of Lemma 15.

Lemma 17. *The machines $M_2^{(3)}$ and $M_2^{(4)}$ are equivalent with respect to I.*

Next we create $M_2^{(5)}$ from $M_2^{(4)}$ by adding a buffer machine B_{SC}^* to $M_2^{(4)}$, i.e. $M_2^{(5)} = M_2^{(4)} + B_{SC}^*$. The proof of the following lemma corresponds to the proof of Lemma 16.

Lemma 18. *The machines $M_2^{(4)}$ and $M_2^{(5)}$ 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

$$(s_5, -) \quad (s_5, -) \quad (\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Register.Ok})$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{lll} (s_4^{(1)}, -) & (s_5, -) & (\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Register.Ok}) \\ (s_5, -) & & (P_i, B^*_{SM}, \mathbf{B.Add}, P_i). \end{array}$$

We note that $((s_5, pk_i^s, pk_i^r), (s_4^{(1)}, -), (s_5, -))$ 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 $((s_5, pk_i^s, pk_i^r), (s_4^{(1)}, -), (s_5, -))$ 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 $((s_5, pk_i^s, pk_i^r), (s_4^{(1)}, -), (s_5, -))$. Let the function $h : S_0 \to S_1^{(6)} \setminus S_0$ be defined as follows: h maps s to the same state, except that for each $i \in I$, $C_1(i)$ is in state $((s_5, pk_i^s, pk_i^r), (s_5, -), (s_5, -))$. It is clear that h is well-defined.

Furthermore, assume that s' is a state in S_0 such that $(s'', w, s') \in f_1^{(6)}$ 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 $((s_5, pk_i^s, pk_i^r), (s_4^{(1)}, -), (s_5, -))$, since this is not a wait state. Let the partial function $g : S_0 \to I_1^{(6)}$ be defined as follows: g maps each such s' to the message $(P_i, B^*_{SM}, \mathbf{B.Add}, P_i)$, 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 $\mathbf{B.Add}$ meant for B^*_{SM} are included in $I^{(5)}$, $M_1^{(5)}$ 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 $\tau : L(M_1^{(6)}) \to L(M_1^{(5)})$. Since τ 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

$$(s_7, P_j, m) \quad (\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{ll} (s_6^{(1)}, P_j, m) & (\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m) \\ (s_7, P_j, m) & (P_i, P_i, \mathbf{SM.Encrypt}, P_j, |m|). \end{array}$$

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

$$(s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) \quad (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Ciphertext}, pk_i^s, pk_j^r, c)$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{ll}(s_{10}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c) & (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Ciphertext}, pk_i^s, pk_j^r, c)\\ (s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) & (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \textbf{SM.Ciphertext}, P_i, P_j, c).\end{array}$$

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

$$(s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j) \quad (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Plaintext}, pk_j^s, pk_i^r, m')$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{ll}(s_{14}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m') & (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Plaintext}, pk_j^s, pk_i^r, m')\\ (s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j) & (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \textbf{SM.Plaintext}, P_i, P_j, m').\end{array}$$

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

$$(s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) \quad (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \textbf{SM.Ciphertext}, P_i, P_j, c)$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{ll}(s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c) & (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \textbf{SM.Ciphertext}, P_i, P_j, c)\\ (s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) & (\mathcal{F}_{SC}, B_{SM}^*, \textbf{B.Lookup}, P_j, P_j/\perp).\end{array}$$

Furthermore, we replace the line

$$(s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j) \quad (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \textbf{SM.Plaintext}, P_i, P_j, m')$$

by the following two lines:

$$\begin{array}{ll}(s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m') & (\mathcal{F}_{SC}, \mathcal{F}_{SC}, \textbf{SM.Plaintext}, P_i, P_j, m')\\ (s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j) & (\mathcal{F}_{SC}, B_{SM}^*, \textbf{B.Lookup}, P_j, P_j/\perp).\end{array}$$

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 $(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, \perp)$ 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 $((s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m'), (s_{17}, -), (s_5, -))$ 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 $((s_{14}^{(2)}, pk_{i'}^s, pk_{i'}^r, P_{j'}, pk_{j'}^s, c_{i'}, m'_{i'}), (s_{17}, -), (s_5, -))$. Let the function $h : S_0 \to S_1^{(10)} \backslash S_0$ 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 $((s_{15}, pk_{i'}^s, pk_{i'}^r, pk_{j'}^s, c_{i'}, m'_{i'}, P_{j'}), (s_{17}, -), (s_5, -))$. 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 $((s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m'), (s_{17}, -), (s_5, -))$, since these are not wait states. Let the partial function $g : S_0 \to I_1^{(10)}$ be defined as follows: g maps each such s' to the message $(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, P_j)$ if P_j is contained in the state of B_{SM}^*, and $(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, \perp)$ 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 $\mathbf{B.Lookup}$ 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 $\tau : L(M_1^{(10)}) \to L(M_1^{(9)})$. Since τ 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 $(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, \perp)$ 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 $((s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m'), (s_{17}, -), (s_5, -))$, there must be a tuple $(P_j, (pk_j^s, pk_j^r))$ 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, $(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, \perp)$ 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 $(s, (\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, \perp))$. $\square$

We proceed by modifying $M_1^{(10)}$ into a new machine $M_1^{(11)}$ by replacing the line

$$(s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) \quad (\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, P_j/\bot)$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{ll} (s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c) & (\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, P_j/\bot) \\ (s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j) & (\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Add}, (P_i, P_j, c, m)). \end{array}$$

We note that $((s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c), (s_{11}, -), (s_5, -))$ is not a wait state of $C_1(i)$. 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 $\mathcal{F}_{SC}$ 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

$$(s_{13}, P_j, c) \quad (\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{ll} (s_6^{(2)}, P_j, c) & (\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c) \\ (s_{13}, P_j, c) & (P_i, P_i, \mathbf{SM.Decrypt}, P_j, c). \end{array}$$

We note that $(s_5, pk_i^s, pk_i^r), (s_6^{(2)}, P_j, c), (s_5, -)$ 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

$$(s_{17}, pk_i^s, pk_i^r, pk_j^s, c, P_j) \quad (\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (-, pk_j^s, -), (P_j, pk_j^s, pk_j^r))$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{ll} (s_{15}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, c) & (\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (-, pk_j^s, -), (P_j, pk_j^s, pk_j^r)) \\ (s_{17}, pk_i^s, pk_i^r, pk_j^s, c, P_j) & (\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, (P_j, P_i, c, -), (P_j, P_i, c, m)/\bot). \end{array}$$

We note that $((s_{15}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, c), (s_{17}, -), (s_5, -))$ 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 $((s_{15}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, c), (s_{17}, -), (s_5, -))$ 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 $((s_{15}^{(1)}, pk_i^s, pk_i^r,$

$P_{j_i}, pk^s_{j_i}, c_i), (s_{17}, -), (s_5, -))$. Let the function $h : S_0 \to S_1^{(14)} \setminus S_0$ be defined as follows: h maps s to the same state, except that for each $i \in I$, $C_1(i)$ is in state $((s_{17}, pk^s_i, pk^r_i, P_{j_i}, pk^s_{j_i}, c_i), (s_{17}, -), (s_5, -))$. 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 $((s_{15}^{(1)}, pk^s_i, pk^r_i, P_j, pk^s_j, c_i), (s_{17}, -), (s_5, -))$, since this is not a wait state. Let the partial function $g : S_0 \to I_1^{(14)}$ be defined as follows: g maps each such s' to the message $(\mathcal{F}_{SC}, B^*_{SM}, \mathbf{B}.\mathbf{Lookup}, (P_j, P_i, c, -), (P_j, P_i, c, m))$ if an entry (P_j, P_i, c, m) is contained in the state of B^*_{SM}, and $(\mathcal{F}_{SC}, B^*_{SM}, \mathbf{B}.\mathbf{Lookup}, (P_j, P_i, c, -), \perp)$ 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 $\mathbf{B}.\mathbf{Lookup}$ 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 $\tau : L(M_1^{(14)}) \to L(M_1^{(13)})$. Since τ 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

$$\begin{array}{llll} (s_5, pk^s_i, pk^r_i) & (s_8, m) & (s_8, P_j) & \\ & & (s_{10}, -) & (\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B}.\mathbf{Lookup}, (P_j, -), (P_j, \perp)) \end{array}$$

in the description of $C_1(i)$ by the following three lines:

$$\begin{array}{llll} (s_5, pk^s_i, pk^r_i) & (s_8, m) & (s_8, P_j) & \\ & & (s_8^{(1)}, P_j) & (\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B}.\mathbf{Lookup}, (P_j, -), (P_j, \perp)) \\ & & (s_{10}, -) & (\mathcal{F}_{PKI}, \mathcal{F}_{PKI}, \mathbf{SM}.\mathbf{Ciphertext}, P_i, P_j, \perp). \end{array}$$

We note that $((s_5, pk^s_i, pk^r_i), (s_8, m), (s_8^{(1)}, P_j))$ 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

$$\begin{array}{llll} (s_5, pk^s_i, pk^r_i) & (s_{14}, c) & (s_8, P_j) & \\ & & (s_{10}, -) & (\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B}.\mathbf{Lookup}, (P_j, -), (P_j, \perp)) \end{array}$$

in the description of $C_1(i)$ by the following three lines:

$$\begin{array}{llll} (s_5, pk^s_i, pk^r_i) & (s_{14}, c) & (s_8, P_j) & \\ & & (s_8^{(1)}, P_j) & (\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B}.\mathbf{Lookup}, (P_j, -), (P_j, \perp)) \\ & & (s_{10}, -) & (\mathcal{F}_{PKI}, \mathcal{F}_{PKI}, \mathbf{SM}.\mathbf{Plaintext}, P_i, P_j, \perp). \end{array}$$

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

$$(s_{10}, -) \quad (\mathcal{F}_{PKI}, \mathcal{F}_{PKI}, \mathbf{SM.Ciphertext}, P_i, P_j, \perp)$$

in the description of $C_1(i)$ by the following two lines:

$$\begin{array}{ll} (s_8^{(2)}, P_j) & (\mathcal{F}_{PKI}, \mathcal{F}_{PKI}, \mathbf{SM.Ciphertext}, P_i, P_j, \perp) \\ (s_{10}, -) & \quad (\mathcal{F}_{SM}, B_{SM}^*, \mathbf{B.Lookup}, P_j, P_j/\perp). \end{array}$$

Furthermore, we replace the line

$$(s_{10}, -) \quad (\mathcal{F}_{PKI}, \mathcal{F}_{PKI}, \mathbf{SM.Plaintext}, P_i, P_j, \perp)$$

by the following two lines:

$$\begin{array}{ll} (s_8^{(2)}, P_j) & (\mathcal{F}_{PKI}, \mathcal{F}_{PKI}, \mathbf{SM.Plaintext}, P_i, P_j, \perp) \\ (s_{10}, -) & \quad (\mathcal{F}_{SM}, B_{SM}^*, \mathbf{B.Lookup}, P_j, P_j/\perp). \end{array}$$

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 $(\mathcal{F}_{SM}, B_{SM}^*, \mathbf{B.Lookup}, P_j, P_j)$ 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

$$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathbf{SC.Key}, P_i, pk_i^s, pk_i^r) \quad (s_3, pk_i^s, pk_i^r)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll} (\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathbf{SC.Key}, P_i, pk_i^s, pk_i^r) & (s_2^{(1)}, pk_i^s, pk_i^r) \\ (\mathcal{S}, B_{SC}^*, \mathbf{B.Add}, (P_i, pk_i^s, pk_i^r)) & (s_3, pk_i^s, pk_i^r). \end{array}$$

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 $\mathcal{S}$ ensures that public keys are unique, we can restrict the set of states of the buffers B_{SC}^* and $B_{\mathcal{S}}$ 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

$$(\mathcal{S}, B^*_{SC}, \mathbf{B}.\mathbf{Add}, (P_i, pk_i^s, pk_i^r)) \quad (s_3, pk_i^s, pk_i^r)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{cc} (\mathcal{S}, B^*_{SC}, \mathbf{B}.\mathbf{Add}, (P_i, pk_i^s, pk_i^r)) & (s_2^{(2)}, pk_i^s, pk_i^r) \\ (\mathcal{S}, \mathcal{S}, \mathbf{SC}.\mathbf{Key}, P_i, pk_i^s, pk_i^r) & (s_3, pk_i^s, pk_i^r). \end{array}$$

We note that $((s_2, -), (s_2^{(2)}, pk_i^s, pk_i^r))$ 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

$$(\mathcal{S}, \mathcal{S}, \mathbf{SC}.\mathbf{Key}, pk_i^s, pk_i^r) \quad (s_3, pk_i^s, pk_i^r)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{cc} (\mathcal{S}, \mathcal{S}, \mathbf{SC}.\mathbf{Key}, pk_i^s, pk_i^r) & (s_2^{(3)}, pk_i^s, pk_i^r) \\ (\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Register}, P_i, (pk_i^s, pk_i^r)) & (s_3, pk_i^s, pk_i^r). \end{array}$$

We note that $((s_2, -), (s_2^{(3)}, pk_i^s, pk_i^r))$ 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

$$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM}.\mathbf{Encrypt}, P_i, P_j, |m|) \quad (s_7, P_j, m) \quad (s_8, pk_i^s, pk_i^r, P_j, |m|)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ccc} (\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM}.\mathbf{Encrypt}, P_i, P_j, |m|) & (s_7, P_j, m) & (s_7^{(1)}, pk_i^s, pk_i^r, P_j, |m|) \\ (\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Retrieve}, P_j) & & (s_8, pk_i^s, pk_i^r, P_j, |m|). \end{array}$$

Furthermore, we replace the line

$$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM}.\mathbf{Decrypt}, P_i, P_j, c) \quad (s_{13}, P_j, c) \quad (s_{15}, pk_i^s, pk_i^r, P_j, c)$$

by the following two lines:

$$\begin{array}{ccc} (\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM}.\mathbf{Decrypt}, P_i, P_j, c) & (s_{13}, P_j, c) & (s_7^{(2)}, pk_i^s, pk_i^r, P_j, c) \\ (\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Retrieve}, P_j) & & (s_{15}, pk_i^s, pk_i^r, P_j, c). \end{array}$$

We note that $((s_7, P_j, m), (s_7^{(1)}, pk_i^s, pk_i^r, P_j, |m|))$ and $((s_{13}, P_j, c), (s_7^{(2)}, pk_i^s, pk_i^r, P_j, c))$ 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

$$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B}.\mathbf{Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r))) \quad (s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll}(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B}.\mathbf{Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r))) & (s_{10}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, pk_j^r, |m|) \\ (\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Retrieve}.\mathbf{Ok}, (pk_j^s, pk_j^r)) & (s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|).\end{array}$$

Furthermore, we replace the line

$$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B}.\mathbf{Lookup}, (P_j, -, -), (P_j, pk_j^s, pk_j^r)) \quad (s_{19}, pk_i^s, pk_i^r, P_j, pk_j^s, c)$$

by the following two lines:

$$\begin{array}{ll}(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B}.\mathbf{Lookup}, (P_j, -, -), (P_j, pk_j^s, pk_j^r)) & (s_{17}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, pk_j^r, c) \\ (\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Retrieve}.\mathbf{Ok}, (pk_j^s, pk_j^r)) & (s_{19}, pk_i^s, pk_i^r, P_j, pk_j^s, c).\end{array}$$

We note that $((s_7, P_j, m), (s_{10}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, pk_j^r, |m|))$ and $((s_{13}, P_j, c), (s_{17}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, pk_j^r, c))$ 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

$$(\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Retrieve}.\mathbf{Ok}, P_i, (pk_j^s, pk_j^r)) \quad (s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll}(\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Retrieve}.\mathbf{Ok}, P_i, (pk_j^s, pk_j^r)) & (s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|) \\ (\mathcal{S}, \mathcal{S}, \mathbf{SC}.\mathbf{Encrypt}, P_i, pk_j^r, \hat{m}) & (s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|).\end{array}$$

We note that $((s_7, P_j, m), (s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|))$ 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

$$(\mathcal{S}, \mathcal{S}, \mathbf{SC}.\mathbf{Encrypt}, P_i, pk_j^r, m) \quad (s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll}(\mathcal{S}, \mathcal{S}, \mathbf{SC}.\mathbf{Encrypt}, P_i, pk_j^r, m) & (s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|) \\ (\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), (P_k, pk_k^s, pk_j^r)/\bot) & (s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|).\end{array}$$

We note that $((s_7, P_j, m), (s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|))$ 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 $(\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), \bot)$ is not significant, and $(\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), (P_k, pk_k^s, pk_j^r))$ is significant only if $P_k = P_j$ and $pk_k^s = pk_j^s$.*

Proof. Using the construction in Lemma 2, we modify $M_2^{(13)}$ by removing the state $((s_7, P_j, m), (s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|))$ 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 $((s_7, P_{j_i}, m_i),$

$(s_{10}^{(3)}, pk_i^s, pk_i^r, P_{j_i}, pk_{j_i}^r, |m_i|))$. 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 $((s_7, P_{j_i}, m_i), (s_{12}, pk_i^s, pk_i^r, P_{j_i}, pk_{j_i}^r, |m_i|))$. 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 $((s_7, P_j, m), (s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|))$, 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 $(\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), (P_k, pk_k^s, pk_j^r))$ if there is a triple (P_k, pk_k^s, pk_j^r) in the state of B_{SC}^*, and $(\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), \perp)$ 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_j^r 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 $\mathbf{B}.\mathbf{Lookup}$ 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 τ 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 $(\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), \perp)$ is not significant, and that $(\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), (P_k, pk_k^s, pk_j^r))$ is significant only if $P_k = P_j$ and $pk_k^s = pk_j^s$. By tracing backwards, we see that for $M_2^{(13)}$ to reach a state s such that $C_2(i)$ is in state $((s_7, P_j, m), (s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, |m|))$, there must be a pair $(P_j, (pk_j^s, pk_j^r))$ in the state of $B_{\mathcal{S}}$, which is true only if B_{SC}^* contains the triple (P_j, pk_j^s, pk_j^r). Since keys are assumed to be unique, this is the only triple containing pk_j^r in the state of B_{SC}^*. This means that, when $M_2^{(13)}$ is in state s, the message $(\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), \perp)$ is not recognized by B_{SM}^*, and $(\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Lookup}, (-, -, pk_j^r), (P_k, pk_k^s, pk_j^r))$ can be recognized only if $P_k = P_j$ and $pk_k^s = pk_j^s$. □

Next we modify $M_2^{(13)}$ into a new machine $M_2^{(14)}$ by replacing the line

$$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B}.\mathbf{Add}, (P_i, P_j, c, m)) \quad (s_{11}, c)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll} (\mathcal{F}_{SM}, B_{SM}, \mathbf{B}.\mathbf{Add}, (P_i, P_j, c, m)) & (s_{10}^{(1)}, P_j, pk_j^r, m, c) \\ (\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B}.\mathbf{Add}, (pk_i^s, pk_j^r, c, m)) & (s_{11}, c). \end{array}$$

We note that $((s_{10}^{(1)}, P_j, pk_j^r, m, c), (s_7, pk_i^s, pk_i^r))$ 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 $\mathcal{F}_{SM}$ 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

$$(\mathcal{F}_{SM}, B^*_{SC}, \mathbf{B}.\mathbf{Add}, (pk^s_i, pk^r_j, c, m)) \quad (s_{11}, c)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll}(\mathcal{F}_{SM}, B^*_{SC}, \mathbf{B}.\mathbf{Add}, (pk^s_i, pk^r_j, c, m)) & (s^{(2)}_{10}, c) \\ (\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{SC}.\mathbf{Ciphertext}, P_i, c) & (s_{11}, c).\end{array}$$

We note that $((s^{(2)}_{10}, c), (s_7, pk^s_i, pk^r_i))$ 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

$$(\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Retrieve}.\mathbf{Ok}, P_i, (pk^s_j, pk^r_j)) \quad (s_{19}, pk^s_i, pk^r_i, P_j, pk^s_j, c)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll}(\mathcal{S}, \mathcal{S}, \mathbf{PKI}.\mathbf{Retrieve}.\mathbf{Ok}, P_i, (pk^s_j, pk^r_j)) & (s^{(2)}_{17}, pk^s_i, pk^r_i, P_j, pk^s_j, c) \\ (\mathcal{S}, \mathcal{S}, \mathbf{SC}.\mathbf{Decrypt}, P_i, pk^s_j, c) & (s_{19}, pk^s_i, pk^r_i, P_j, pk^s_j, c).\end{array}$$

We note that $((s_{13}, P_j, c), (s^{(2)}_{17}, pk^s_i, pk^r_i, P_j, pk^s_j, c))$ 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

$$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B}.\mathbf{Lookup}, P_j, P_j) \quad (s_{16}, P_j, c, pk^s_j)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll}(\mathcal{F}_{SM}, B_{SM}, \mathbf{B}.\mathbf{Lookup}, P_j, P_j) & (s^{(1)}_{14}, P_j, pk^s_j, c, m') \\ (\mathcal{F}_{SM}, B^*_{SC}, \mathbf{B}.\mathbf{Lookup}, (-, pk^s_j, -), (P_k, pk^s_j, pk^r_k)/\bot) & (s_{16}, P_j, c, pk^s_j).\end{array}$$

We note that $((s^{(1)}_{14}, P_j, c, m'), (s_7, pk^s_i, pk^r_i))$ 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 $(\mathcal{F}_{SM}, B^*_{SC}, \mathbf{B}.\mathbf{Lookup}, (-, pk^s_j, -), \bot)$ is not significant, and $(\mathcal{F}_{SM}, B^*_{SC}, \mathbf{B}.\mathbf{Lookup}, (-, pk^s_j, -), (P_k, pk^s_j, pk^r_k))$ is significant only if $P_k = P_j$ and $pk^r_k = pk^r_j$.*

We proceed by modifying $M_2^{(18)}$ into a new machine $M_2^{(19)}$ by replacing the line

$$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, (P_j, P_i, c, -), \perp/(P_j, P_i, c, m)) \quad (s_{17}, \perp)/(s_{18}, m)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll} (\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, (P_i, P_j, c, -), \perp/(P_j, P_i, c, m)) & (s_{16}^{(1)}, P_j, pk_j^s, c) \\ (\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B.Lookup}, (pk_j^s, pk_i^r, c, -), \perp/(pk_j^s, pk_i^r, c, m)) & (s_{17}, \perp)/(s_{18}, m). \end{array}$$

We note that $((s_{16}^{(1)}, P_j, pk_j^s, c), (s_7, pk_i^s, pk_i^r))$ is not a wait state of $C_2(i)$.

Lemma 25. *The machines* $M_2^{(18)}$ *and* $M_2^{(19)}$ *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 $M_2^{(19)}$ by removing the state $((s_{16}^{(1)}, P_j, pk_j^s, c), (s_7, pk_i^s, pk_i^r))$ and the transition out of it from $C_2(i)$, such that the resulting machine is $M_2^{(18)}$. Let S_0 be the set of states of $M_2^{(19)}$ 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 $((s_{16}^{(1)}, P_{j_i}, pk_{j_i}^s, c_i), (s_7, pk_i^s, pk_i^r))$. Let the function $h : S_0 \to S_2^{(19)} \setminus S_0$ be defined as follows: h maps s to the same state, except that for each $i \in I$, $C_2(i)$ is in state $((s_{18}, m_i), (s_7, pk_i^s, pk_i^r))$ if there is a tuple (P_{j_i}, P_i, c_i, m_i) in the state of B_{SM}, and $((s_{17}, \perp), (s_7, pk_i^s, pk_i^r))$ 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 $(s'', w, s') \in f_2^{(19)}$ for some w in $I_2^{(19)}$, some reachable s'' in $S_2^{(19)}$. Note that s' corresponds to exactly one $C_2(i)$ being in state $((s_{16}^{(1)}, P_j, pk_j^s, c), (s_7, pk_i^s, pk_i^r))$, since this is not a wait state. Let the partial function $g : S_0 \to I_2^{(19)}$ be defined as follows: g maps each such s' to the message $(\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B.Lookup}, (pk_j^s, pk_i^r, c, -), (pk_j^s, pk_i^r, c, m))$ if there is a tuple (pk_j^s, pk_i^r, c, m) in the state of B_{SC}^*, and $(\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B.Lookup}, (pk_j^s, pk_i^r, c, -), \perp)$ 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 $M_2^{(19)}$ 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_j^s, pk_i^r, 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, $g(s') = (\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B.Lookup}, (pk_j^s, pk_i^r, c, -), (pk_j^s, pk_i^r, c, m))$ implies that $h(s') = ((s_{18}, m), (s_7, pk_i^s, pk_i^r))$, while $g(s') = (\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B.Lookup}, (pk_j^s, pk_i^r, c, -), \perp)$ implies that $h(s') = ((s_{17}, \perp), (s_7, pk_i^s, pk_i^r))$. 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 $M_2^{(19)}$

can be modified using Lemma 2. Since messages containing **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 τ 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

$$(\mathcal{F}_{SM}, B^*_{SC}, \mathbf{B.Lookup}, (pk^s_j, pk^r_i, c, -), \perp/(pk^s_j, pk^r_i, c, m)) \quad (s_{17}, \perp)/(s_{18}, m).$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{rl}(\mathcal{F}_{SM}, B^*_{SC}, \mathbf{B.Lookup}, (pk^s_j, pk^r_i, c, -), \perp/(pk^s_j, pk^r_i, c, m)) & (s^{(2)}_{16}, \perp)/(s^{(3)}_{16}, m)\\ (\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{SC.Plaintext}, P_i, \perp/m) & (s_{17}, \perp)/(s_{18}, m).\end{array}$$

We note that $((s^{(2)}_{16}, \perp), (s_7, pk^s_i, pk^r_i))$ and $((s^{(3)}_{16}, m), (s_7, pk^s_i, pk^r_i))$ 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 $((s^{(2)}_{16}, \perp), (s_7, pk^s_i, pk^r_i))$ and $((s^{(3)}_{16}, m), (s_7, pk^s_i, pk^r_i))$ 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 $((s^{(2)}_{16}, \perp), (s_7, pk^s_i, pk^r_i))$, and for each i' in some index set I', $C_2(i')$ is in state $((s^{(3)}_{16}, m_{i'}), (s_7, pk^s_{i'}, pk^r_{i'}))$ 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 $((s_{17}, \perp), (s_7, pk^s_i, pk^r_i))$, and for each $i' \in I'$, $C_2(i')$ is in state $((s_{18}, m_{i'}), (s_7, pk^s_{i'}, pk^r_{i'}))$. It is clear that h is well-defined.

Furthermore, assume that s' is a state in S_0 such that $(s'', w, s') \in f_2^{(20)}$ 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 $((s^{(2)}_{16}, \perp), (s_7, pk^s_i, pk^r_i))$ or $((s^{(3)}_{16}, m), (s_7, pk^s_i, pk^r_i))$, 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 $(\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{SC.Plaintext}, \perp)$ if $C_2(i)$ is in state $((s^{(2)}_{16}, \perp), (s_7, pk^s_i, pk^r_i))$ and $(\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{SC.Plaintext}, m)$ if $C_2(i)$ is in state $((s^{(3)}_{16}, m), (s_7, pk^s_i, pk^r_i))$. 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 $(\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{SC.Plaintext}, \perp/m)$, 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 τ 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

$$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, \perp) \quad (s_9, \perp)$$

in the description of $C_2(i)$ by the following two lines:

$$\begin{array}{ll} (\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, \perp) & (s_8^{(1)}, m, \perp) \\ (\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{PKI.Retrieve.Fail}, P_i) & (s_9, \perp). \end{array}$$

Furthermore, the line

$$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, \perp) \quad (s_{15}, \perp)$$

is replaced by the following two lines:

$$\begin{array}{ll} (\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, \perp) & (s_{14}^{(2)}, c, \perp) \\ (\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{PKI.Retrieve.Fail}, P_i) & (s_{15}, \perp). \end{array}$$

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 (σ, τ) between the two machines, such that τ 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 τ, 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 τ is a bijection.

We proceed by describing the map σ. Recall that S_1 and S_2 are given by

$$S_1 = S_{B_{SC}} \times S_{B_{PKI}} \times S_{B_{SM}^*} \times \prod_{i=1}^{n} S_{C_1(i)},$$

$$S_2 = S_{B_{SC}^*} \times S_{B_{\mathcal{S}}} \times S_{B_{SM}} \times \prod_{i=1}^{n} S_{C_2(i)}.$$

It will be convenient to describe σ using the maps

$$\begin{aligned}
&\sigma_{B^*_{SC}} : S_{B_{SC}} \to S_{B^*_{SC}},\\
&\sigma_{B_{\mathcal{S}}} : S_{B_{PKI}} \to S_{B_{\mathcal{S}}},\\
&\sigma_{B_{SM}} : S_{B^*_{SM}} \to S_{B_{SM}},\\
&\sigma_i : S_{C_1(i)} \to S_{C_2(i)}.
\end{aligned}$$

That is, for a state s in S_1, $\sigma(s)$ is given by

$$\sigma(s) = (\sigma_{B^*_{SC}}(s_{B_{SC}}), \sigma_{B_{\mathcal{S}}}(s_{B_{PKI}}), \sigma_{B_{SM}}(s_{B^*_{SM}}), \sigma_1(s_{C_1(1)}), \ldots, \sigma_n(s_{C_1(n)})).$$

We now describe the map σ_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 σ_i is a bijection. Another important property of σ_i is preservation of wait states. More precisely, σ_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 $\sigma_{B^*_{SC}}$, $\sigma_{B_{\mathcal{S}}}$ and $\sigma_{B_{SM}}$, which simply copy the states of the respective buffers, that is, for a state s of S^*_1, $\sigma_{B^*_{SC}}(s_{B_{SC}}) = s_{B_{SC}}$, $\sigma_{B_{\mathcal{S}}}(s_{B_{PKI}}) = s_{B_{PKI}}$ and $\sigma_{B_{SM}}(s_{B^*_{SM}}) = s_{B^*_{SM}}$. It is clear that $\sigma_{B^*_{SC}}$, $\sigma_{B_{\mathcal{S}}}$ and $\sigma_{B_{SM}}$ are all bijective. In particular, since every state of $\sigma_{B^*_{SC}}$, $\sigma_{B_{\mathcal{S}}}$ and $\sigma_{B_{SM}}$ is a wait state, each map gives a one-to-one correspondence between wait states.

Given that σ_i, $\sigma_{B^*_{SC}}$, $\sigma_{B_{\mathcal{S}}}$ and $\sigma_{B_{SM}}$ are all bijective, we conclude that $\sigma : S_1 \to S_2$ is a bijection. We clearly have $\sigma(s_1) = s_2$. Furthermore, the fact that σ_i, $\sigma_{B^*_{SC}}$, $\sigma_{B_{\mathcal{S}}}$ 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 $w = (\mathcal{Z}, P_i, \textbf{SM.Register})$. 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 $((s_5, pk^s_i, pk^r_i), (s_6, -), (s_5, -))$, 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 $((s_5, -), (s_7, pk^s_i, pk^r_i))$, 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 $\sigma_i((s_0, -), (s_0, -), (s_0, -)) = ((s_0, -), (s_0, -))$, and $\sigma_i((s_5, pk^s_i, pk^r_i), (s_6, -), (s_5, -)) = ((s_5, -), (s_7, pk^s_i, pk^s_i))$. The one-to-one correspondences between wait states provided by σ then implies that σ 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 $w = (P_i, \mathcal{F}_{SC}, \mathbf{SC}.\mathbf{KeyGen})$. Note that $\tau(w) = (\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM}.\mathbf{Register}, P_i)$. 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 σ then implies that σ 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 $w = (\mathcal{F}_{SC}, B_{SC}, \mathbf{B}.\mathbf{Add}, (P_i, pk_i^s, pk_i^r))$. Note that $\tau(w) = (\mathcal{S}, B_{SC}^*, \mathbf{B}.\mathbf{Add}, (P_i, pk_i^s, pk_i^r))$. 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 σ_i and the one-to-one correspondences between wait states provided by σ to argue that σ 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^s, pk_i^r) 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^s, pk_i^r) 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 $\mathbf{B}.\mathbf{Add}$, the properties in question can be proved by the above procedure, using that σ copies elements from B_{SC} to B_{SC}^*, elements from B_{PKI} to $B_{\mathcal{S}}$, and elements from B_{SM}^* to B_{SM}.

We now consider the message $w = (\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B}.\mathbf{Lookup}, (P_j,-), (P_j, (pk_j^s, pk_j^r)))$. 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 $(P_j, (pk_j^s, pk_j^r))$ in its state. Similarly, for $\tau(w) = (\mathcal{S}, B_{\mathcal{S}}, \mathbf{B}.\mathbf{Lookup}, (P_j,-), (P_j, (pk_j^s, pk_j^r)))$ to be recognized by M_2, it must be recognized both by $M_{\mathcal{S}}(i)$ and the buffer machine

$B_{\mathcal{S}}$. The latter is the case if $B_{\mathcal{S}}$ has a pair $(P_j, (pk^s_j, pk^r_j))$ in its state. Since σ copies elements from B_{PKI} to $B_{\mathcal{S}}$, 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 $B_{\mathcal{S}}$ 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 σ_i and the one-to-one correspondences between wait states provided by σ to argue that σ 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 σ copies elements from B_{SC} to B^*_{SC}, elements from B_{PKI} to $B_{\mathcal{S}}$, and elements from B^*_{SM} to B_{SM}.

This means that $(\sigma, \tau) : M_1^{(18)} \to M_2^{(22)}$ is an isomorphism. By Lemma 1, the induced language map is a bijection, and since τ leaves I fixed, it follows from Lemma 3 that $M_1^{(18)}$ and $M_2^{(22)}$ are equivalent with respect to I. $\square$

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 π_{SM} securely realizes the functionality $\mathcal{F}_{SM}$ in the $(\mathcal{F}_{PKI}, \mathcal{F}_{SC})$-hybrid model. $\square$

References

[1] Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *CCS '97: Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 36–47, New York, NY, USA, 1997. ACM.

[2] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A General Composition Theorem for Secure Reactive System. In *Proceedings of 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, February 2004.

[3] Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In *CRYPTO '93: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, pages 232–249, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[4] Michael Burrows, Martin Abadi, and Roger Needham. A Logic of Authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[5] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Available at `http://eprint.iacr.org/2000/067`.

[6] Ran Canetti. Composable Formal Security Analysis: Juggling Soundness, Simplicity and Efficiency. In *ICALP '08: Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, pages 1–13, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] Ran Canetti and Jonathan Herzog. Universally Composable Symbolic Analysis of Cryptographic Protocols (the Case of Encryption-Based Mutual Authentication and Key Exchange). In *2004/334, International Association for Cryptological Research*, 2004.

[8] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

$M_{SC}(i)$	$M_\pi(i)$	$M_{PKI}(i)$			$M_{SM}(i)$	$M_S(i)$
$(s_0, -)$	$(s_0, -)$	$(s_0, -)$			$(s_0, -)$	$(s_0, -)$
	$(s_1, -)$		$(\mathcal{Z}, P_i, \mathbf{SM.Register})$	$(\mathcal{Z}, P_i, \mathbf{SM.Register})$	$(s_1, -)$	
$(s_1, -)$	$(s_2, -)$		$(P_i, \mathcal{F}_{SC}, \mathbf{SC.KeyGen})$	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Register}, P_i)$	$(s_2, -)$	$(s_1, -)$
$(s_2, -)$			$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \mathbf{SC.KeyGen}, P_i)$	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \mathbf{SC.KeyGen}, P_i)$		$(s_2, -)$
(s_3, pk_i^s, pk_i^r)			$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathbf{SC.Key}, P_i, pk_i^s, pk_i^r)$	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathbf{SC.Key}, P_i, pk_i^s, pk_i^r)$		(s_3, pk_i^s, pk_i^r)
(s_4, pk_i^s, pk_i^r)			$(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Add}, (P_i, pk_i^s, pk_i^r))$			
(s_5, pk_i^s, pk_i^r)	(s_3, pk_i^s, pk_i^r)		$(\mathcal{F}_{SC}, P_i, \mathbf{SC.Key}, pk_i^s, pk_i^r)$			
	$(s_4, -)$	$(s_1, (pk_i^s, pk_i^r))$	$(P_i, \mathcal{F}_{PKI}, \mathbf{PKI.Register}, (pk_i^s, pk_i^r))$			
		$(s_2, (pk_i^s, pk_i^r))$	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \mathbf{PKI.Register}, P_i, (pk_i^s, pk_i^r))$	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \mathbf{PKI.Register}, P_i, (pk_i^s, pk_i^r))$		(s_4, pk_i^s, pk_i^r)
		$(s_3, (pk_i^s, pk_i^r))$	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \mathbf{PKI.Register.Ok}, P_i)$	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \mathbf{PKI.Register.Ok}, P_i)$		(s_5, pk_i^s, pk_i^r)
		$(s_4, -)$	$(\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B.Add}, (P_i, (pk_i^s, pk_i^r)))$	$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Add}, (P_i, (pk_i^s, pk_i^r)))$		(s_6, pk_i^s, pk_i^r)
	$(s_5, -)$	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Register.Ok})$	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Register.Ok}, P_i)$	$(s_3, -)$	(s_7, pk_i^s, pk_i^r)
				$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Add}, P_i)$	$(s_4, -)$	
	$(s_6, -)$		$(P_i, \mathcal{Z}, \mathbf{SM.Register.Ok})$	$(P_i, \mathcal{Z}, \mathbf{SM.Register.Ok})$	$(s_5, -)$	
(s_5, pk_i^s, pk_i^r)	$(s_6, -)$	$(s_5, -)$			$(s_5, -)$	(s_7, pk_i^s, pk_i^r)
	$(s_{20}, -)$		$(\mathcal{Z}, P_i, \mathbf{SM.Register})$	$(\mathcal{Z}, P_i, \mathbf{SM.Register})$	$(s_{20}, -)$	
	$(s_6, -)$		$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(s_5, -)$	

FIGURE 13. Registration

$M_{SC}(i)$	$M_{\pi}(i)$	$M_{PKI}(i)$			$M_{SM}(i)$	$M_{\mathcal{S}}(i)$
$(s_0, -)$	$(s_0, -)$	$(s_0, -)$			$(s_0, -)$	$(s_0, -)$
	$(s_{19}, -)$		$(\mathcal{Z}, P_i, \textbf{SM.Encrypt}, P_j, m)$	$(\mathcal{Z}, P_i, \textbf{SM.Encrypt}, P_j, m)$	$(s_{19}, -)$	
	$(s_0, -)$		$(P_i, \mathcal{Z}, \textbf{SM.Error})$	$(P_i, \mathcal{Z}, \textbf{SM.Error})$	$(s_0, -)$	
(s_5, pk_i^s, pk_i^r)	$(s_6, -)$	$(s_5, -)$			$(s_5, -)$	(s_7, pk_i^s, pk_i^r)
	(s_7, P_j, m)		$(\mathcal{Z}, P_i, \textbf{SM.Encrypt}, P_j, m)$	$(\mathcal{Z}, P_i, \textbf{SM.Encrypt}, P_j, m)$	(s_6, P_j, m)	
				$(\mathcal{F}_{SM}, \mathcal{S}, \textbf{SM.Encrypt}, P_i, P_j, \|m\|)$	(s_7, P_j, m)	$(s_8, pk_i^s, pk_i^r, P_j, \|m\|)$
	(s_8, m)	(s_6, P_j)	$(P_i, \mathcal{F}_{PKI}, \textbf{PKI.Retrieve}, P_j)$			
		(s_7, P_j)	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \textbf{PKI.Retrieve}, P_i, P_j)$	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \textbf{PKI.Retrieve}, P_i, P_j)$		$(s_9, pk_i^s, pk_i^r, P_j, \|m\|)$
		(s_8, P_j)	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \textbf{PKI.Retrieve.Ok}, P_i, P_j)$	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \textbf{PKI.Retrieve.Ok}, P_i, P_j)$		$(s_{10}, pk_i^s, pk_i^r, P_j, \|m\|)$
		$(s_9, (pk_j^s, pk_j^r), P_j)$	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$	$(\mathcal{S}, B_{\mathcal{S}}, \textbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$		$(s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, \|m\|)$
	(s_{10}, pk_j^r, m, P_j)	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI.Retrieve.Ok}, (pk_j^s, pk_j^r))$			
$(s_6, pk_i^s, pk_i^r, pk_j^r, m, P_j)$	$(s_{11}, -)$		$(P_i, \mathcal{F}_{SC}, \textbf{SC.Encrypt}, pk_j^r, m)$			
$(s_9, pk_i^s, pk_i^r, pk_j^r, m, P_j)$			$(\mathcal{F}_{SC}, B_{SC}, \textbf{B.Lookup}, (-, -, pk_j^r), (P_j, pk_j^s, pk_j^r))$			
$(s_{10}, pk_i^s, pk_i^r, pk_j^r, m, P_j)$			$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC.Encrypt}, pk_i^s, pk_j^r, \|m\|)$	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC.Encrypt}, pk_i^s, pk_j^r, \|m\|)$		$(s_{13}, pk_i^s, pk_i^r, P_j, pk_j^r)$
$(s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j)$			$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Ciphertext}, pk_i^s, pk_j^r, c)$	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Ciphertext}, pk_i^s, pk_j^r, c)$		$(s_{14}, pk_i^s, pk_i^r, P_j, c, pk_j^r)$
				$(\mathcal{S}, \mathcal{F}_{SM}, \textbf{SM.Ciphertext}, P_i, P_j, c)$	(s_8, P_j, m, c, pk_j^r)	(s_7, pk_i^s, pk_i^r)
				$(\mathcal{F}_{SM}, B_{SM}, \textbf{B.Lookup}, P_j, P_j)$	$(s_{10}, P_j, m, c, pk_j^r)$	
				$(\mathcal{F}_{SM}, B_{SM}, \textbf{B.Add}, (P_i, P_j, c, m))$	(s_{11}, c)	
$(s_{12}, pk_i^s, pk_i^r, c)$			$(\mathcal{F}_{SC}, B_{SC}, \textbf{B.Add}, (pk_i^s, pk_j^r, c, m))$			
(s_5, pk_i^s, pk_i^r)	(s_{12}, c)		$(\mathcal{F}_{SC}, P_i, \textbf{SC.Ciphertext}, c)$			
	$(s_6, -)$		$(P_i, \mathcal{Z}, \textbf{SM.Ciphertext}, c)$	$(P_i, \mathcal{Z}, \textbf{SM.Ciphertext}, c)$	$(s_5, -)$	

FIGURE 14. Encryption

$M_{SC}(i)$	$M_{\pi}(i)$	$M_{PKI}(i)$			$M_{SM}(i)$	$M_{\mathcal{S}}(i)$
$(s_0, -)$	$(s_0, -)$	$(s_0, -)$			$(s_0, -)$	$(s_0, -)$
	$(s_{19}, -)$		$(\mathcal{Z}, P_i, \textbf{SM.Decrypt}, P_j, c)$	$(\mathcal{Z}, P_i, \textbf{SM.Decrypt}, P_j, c)$	$(s_{19}, -)$	
	$(s_0, -)$		$(P_i, \mathcal{Z}, \textbf{SM.Error})$	$(P_i, \mathcal{Z}, \textbf{SM.Error})$	$(s_0, -)$	
(s_5, pk_i^s, pk_i^r)	$(s_6, -)$	$(s_5, -)$			$(s_5, -)$	(s_7, pk_i^s, pk_i^r)
	(s_{13}, P_j, c)		$(\mathcal{Z}, P_i, \textbf{SM.Decrypt}, P_j, c)$	$(\mathcal{Z}, P_i, \textbf{SM.Decrypt}, P_j, c)$	(s_{12}, P_j, c)	
				$(\mathcal{F}_{SM}, \mathcal{S}, \textbf{SM.Decrypt}, P_i, P_j, c)$	(s_{13}, P_j, c)	$(s_{15}, pk_i^s, pk_i^r, P_j, c)$
	(s_{14}, c)	(s_6, P_j)	$(P_i, \mathcal{F}_{PKI}, \textbf{PKI.Retrieve}, P_j)$			
		(s_7, P_j)	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \textbf{PKI.Retrieve}, P_i, P_j)$	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \textbf{PKI.Retrieve}, P_i, P_j)$		$(s_{16}, pk_i^s, pk_i^r, P_j, c)$
		(s_8, P_j)	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \textbf{PKI.Retrieve.Ok}, P_i, P_j)$	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \textbf{PKI.Retrieve.Ok}, P_i, P_j)$		$(s_{17}, pk_i^s, pk_i^r, P_j, c)$
		$(s_9, (pk_j^s, pk_j^r), P_j)$	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$	$(\mathcal{S}, B_{\mathcal{S}}, \textbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$		$(s_{19}, pk_i^s, pk_i^r, P_j, pk_j^s, c)$
	(s_{16}, pk_j^s, c, P_j)	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI.Retrieve.Ok}, (pk_j^s, pk_j^r))$			
$(s_{13}, pk_i^s, pk_i^r, pk_j^s, c, P_j)$	$(s_{17}, -)$		$(P_i, \mathcal{F}_{SC}, \textbf{SC.Decrypt}, pk_j^s, c)$			
$(s_{14}, pk_i^s, pk_i^r, pk_j^s, c, P_j)$			$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC.Decrypt}, pk_j^s, pk_i^r, c)$	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC.Decrypt}, pk_j^s, pk_i^r, c)$		$(s_{20}, pk_i^s, pk_i^r, P_j, pk_j^s)$
$(s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j)$			$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Plaintext}, pk_j^s, pk_i^r, m')$	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Plaintext}, pk_j^s, pk_i^r, m')$		$(s_{21}, pk_i^s, pk_i^r, P_j, m', pk_j^s)$
				$(\mathcal{S}, \mathcal{F}_{SM}, \textbf{SM.Plaintext}, P_i, P_j, m')$	$(s_{14}, P_j, c, m', pk_j^s)$	(s_7, pk_i^s, pk_i^r)
				$(\mathcal{F}_{SM}, B_{SM}, \textbf{B.Lookup}, P_j, P_j)$	(s_{16}, P_j, c, pk_j^s)	
$(s_{17}, pk_i^s, pk_i^r, pk_j^s, c, P_j)$			$(\mathcal{F}_{SC}, B_{SC}, \textbf{B.Lookup}, (-, pk_j^s, -), (P_j, pk_j^s, pk_j^r))$			
				$(\mathcal{F}_{SM}, B_{SM}, \textbf{B.Lookup}, (P_j, P_i, c, -), \perp/(P_j, P_i, c, m))$	$(s_{17}, \perp)/(s_{18}, m)$	
$(s_{18}, pk_i^s, pk_i^r, m)/(s_{18}, pk_i^s, pk_i^r, \perp)$			$(\mathcal{F}_{SC}, B_{SC}, \textbf{B.Lookup}, (pk_j^s, pk_i^r, c, -), \perp/(pk_j^s, pk_i^r, c, m))$			
(s_5, pk_i^s, pk_i^r)	$(s_{18}, \perp/m)$		$(\mathcal{F}_{SC}, P_i, \textbf{SC.Plaintext}, \perp/m)$			
	$(s_6, -)$		$(P_i, \mathcal{Z}, \textbf{SM.Plaintext}, \perp/m)$	$(P_i, \mathcal{Z}, \textbf{SM.Plaintext}, \perp/m)$	$(s_5, -)$	

FIGURE 15. Decryption

$M_{SC}(i)$	$M_\pi(i)$	$M_{PKI}(i)$			$M_{SM}(i)$	$M_{\mathcal{S}}(i)$
(s_5, pk_i^s, pk_i^r)	(s_8, m)	(s_8, P_j)			(s_7, P_j, m)	$(s_{10}, pk_i^s, pk_i^r, P_j, \|m\|)$
		$(s_{10}, -)$	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B}.\textbf{Lookup}, (P_j, -), (P_j, \bot))$	$(\mathcal{S}, B_{\mathcal{S}}, \textbf{B}.\textbf{Lookup}, (P_j, -), (P_j, \bot))$		$(s_{11}, pk_i^s, pk_i^r, P_j)$
				$(\mathcal{S}, \mathcal{F}_{SM}, \textbf{SM}.\textbf{Ciphertext}, P_i, P_j, \bot)$	$(s_8, P_j, m, \bot)$	(s_7, pk_i^s, pk_i^r)
				$(\mathcal{F}_{SM}, B_{SM}, \textbf{B}.\textbf{Lookup}, P_j, \bot)$	$(s_9, \bot)$	
	$(s_9, -)$	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI}.\textbf{Retrieve}.\textbf{Fail})$			
	$(s_6, -)$		$(P_i, \mathcal{Z}, \textbf{SM}.\textbf{Not}.\textbf{Registered})$	$(P_i, \mathcal{Z}, \textbf{SM}.\textbf{Not}.\textbf{Registered})$	$(s_5, -)$	
(s_5, pk_i^s, pk_i^r)	(s_{14}, c)	(s_8, P_j)			(s_{13}, P_j, c)	$(s_{17}, pk_i^s, pk_i^r, P_j, c)$
		$(s_{10}, -)$	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B}.\textbf{Lookup}, (P_j, -), (P_j, \bot))$	$(\mathcal{S}, B_{\mathcal{S}}, \textbf{B}.\textbf{Lookup}, (P_j, -), (P_j, \bot))$		$(s_{16}, pk_i^s, pk_i^r, P_j)$
				$(\mathcal{S}, \mathcal{F}_{SM}, \textbf{SM}.\textbf{Plaintext}, P_i, P_j, \bot)$	$(s_{14}, P_j, c, \bot)$	(s_7, pk_i^s, pk_i^r)
				$(\mathcal{F}_{SM}, B_{SM}, \textbf{B}.\textbf{Lookup}, P_j, \bot)$	$(s_{15}, \bot)$	
	$(s_{15}, -)$	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI}.\textbf{Retrieve}.\textbf{Fail})$			
	$(s_6, -)$		$(P_i, \mathcal{Z}, \textbf{SM}.\textbf{Not}.\textbf{Registered})$	$(P_i, \mathcal{Z}, \textbf{SM}.\textbf{Not}.\textbf{Registered})$	$(s_5, -)$	

FIGURE 16. Special cases during encryption and decryption

$M_{SC}(i)$	$M_\pi(i)$	$M_{PKI}(i)$			$M_{SM}(i)$	$M_{\mathcal{S}}(i)$
$(s_0, -)$	$(s_0, -)$	$(s_0, -)$			$(s_0, -)$	$(s_0, -)$
	$(s_1, -)$		$(\mathcal{Z}, P_i, \textbf{SM.Register})$	$(\mathcal{Z}, P_i, \textbf{SM.Register})$	$(s_1, -)$	
$(s_1, -)$	$(s_2, -)$		$(P_i, \mathcal{F}_{SC}, \textbf{SC.KeyGen})$	$(\mathcal{F}_{SM}, \mathcal{S}, \textbf{SM.Register}, P_i)$	$(s_2, -)$	$(s_1, -)$
$(s_2, -)$			$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC.KeyGen}, P_i)$	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \textbf{SC.KeyGen}, P_i)$		$(s_2, -)$
(s_3, pk_i^s, pk_i^r)			$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Key}, P_i, pk_i^s, pk_i^r)$	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \textbf{SC.Key}, P_i, pk_i^s, pk_i^r)$		$(s_2^{(1)}, pk_i^s, pk_i^r)$
(s_4, pk_i^s, pk_i^r)			$(\mathcal{F}_{SC}, B_{SC}, \textbf{B.Add}, (P_i, pk_i^s, pk_i^r))$	$(\mathcal{S}, B^*_{SC}, \textbf{B.Add}, (P_i, pk_i^s, pk_i^r))$		$(s_2^{(2)}, pk_i^s, pk_i^r)$
(s_5, pk_i^s, pk_i^r)	(s_3, pk_i^s, pk_i^r)		$(\mathcal{F}_{SC}, P_i, \textbf{SC.Key}, pk_i^s, pk_i^r)$	$(\mathcal{S}, \mathcal{S}, \textbf{SC.Key}, P_i, pk_i^s, pk_i^r)$		$(s_2^{(3)}, pk_i^s, pk_i^r)$
	$(s_4, -)$	$(s_1, (pk_i^s, pk_i^r))$	$(P_i, \mathcal{F}_{PKI}, \textbf{PKI.Register}, (pk_i^s, pk_i^r))$	$(\mathcal{S}, \mathcal{S}, \textbf{PKI.Register}, P_i, (pk_i^s, pk_i^r))$		(s_3, pk_i^s, pk_i^r)
		$(s_2, (pk_i^s, pk_i^r))$	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \textbf{PKI.Register}, P_i, (pk_i^s, pk_i^r))$	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \textbf{PKI.Register}, P_i, (pk_i^s, pk_i^r))$		(s_4, pk_i^s, pk_i^r)
		$(s_3, (pk_i^s, pk_i^r))$	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \textbf{PKI.Register.Ok}, P_i)$	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \textbf{PKI.Register.Ok}, P_i)$		(s_5, pk_i^s, pk_i^r)
		$(s_4, -)$	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B.Add}, (P_i, (pk_i^s, pk_i^r)))$	$(\mathcal{S}, B_{\mathcal{S}}, \textbf{B.Add}, (P_i, (pk_i^s, pk_i^r)))$		(s_6, pk_i^s, pk_i^r)
	$(s_4^{(1)}, -)$	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI.Register.Ok})$	$(\mathcal{S}, \mathcal{F}_{SM}, \textbf{SM.Register.Ok}, P_i)$	$(s_3, -)$	(s_7, pk_i^s, pk_i^r)
	$(s_5, -)$		$(P_i, B^*_{SM}, \textbf{B.Add}, P_i)$	$(\mathcal{F}_{SM}, B_{SM}, \textbf{B.Add}, P_i)$	$(s_4, -)$	
	$(s_6, -)$		$(P_i, \mathcal{Z}, \textbf{SM.Register.Ok})$	$(P_i, \mathcal{Z}, \textbf{SM.Register.Ok})$	$(s_5, -)$	
(s_5, pk_i^s, pk_i^r)	$(s_6, -)$	$(s_5, -)$			$(s_5, -)$	(s_7, pk_i^s, pk_i^r)
	$(s_{20}, -)$		$(\mathcal{Z}, P_i, \textbf{SM.Register})$	$(\mathcal{Z}, P_i, \textbf{SM.Register})$	$(s_{20}, -)$	
	$(s_6, -)$		$(P_i, \mathcal{Z}, \textbf{SM.Error})$	$(P_i, \mathcal{Z}, \textbf{SM.Error})$	$(s_5, -)$	

FIGURE 17. Modified registration

$M_{SC}(i)$	$M_\pi(i)$	$M_{PKI}(i)$			$M_{SM}(i)$	$M_{\mathcal{S}}(i)$
$(s_0, -)$	$(s_0, -)$	$(s_0, -)$			$(s_0, -)$	$(s_0, -)$
	$(s_{19}, -)$		$(\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$	$(\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$	$(s_{19}, -)$	
	$(s_0, -)$		$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(s_0, -)$	
(s_5, pk_i^s, pk_i^r)	$(s_6, -)$	$(s_5, -)$			$(s_5, -)$	(s_7, pk_i^s, pk_i^r)
	$(s_6^{(1)}, P_j, m)$		$(\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$	$(\mathcal{Z}, P_i, \mathbf{SM.Encrypt}, P_j, m)$	(s_6, P_j, m)	
	(s_7, P_j, m)		$(P_i, P_i, \mathbf{SM.Encrypt}, P_j, \lvert m \rvert)$	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Encrypt}, P_i, P_j, \lvert m \rvert)$	(s_7, P_j, m)	$(s_7^{(1)}, pk_i^s, pk_i^r, P_j, \lvert m \rvert)$
	(s_8, m)	(s_6, P_j)	$(P_i, \mathcal{F}_{PKI}, \mathbf{PKI.Retrieve}, P_j)$	$(\mathcal{S}, \mathcal{S}, \mathbf{PKI.Retrieve}, P_i, P_j)$		$(s_8, pk_i^s, pk_i^r, P_j, \lvert m \rvert)$
		(s_7, P_j)	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \mathbf{PKI.Retrieve}, P_i, P_j)$	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \mathbf{PKI.Retrieve}, P_i, P_j)$		$(s_9, pk_i^s, pk_i^r, P_j, \lvert m \rvert)$
		(s_8, P_j)	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \mathbf{PKI.Retrieve.Ok}, P_i, P_j)$	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \mathbf{PKI.Retrieve.Ok}, P_i, P_j)$		$(s_{10}, pk_i^s, pk_i^r, P_j, \lvert m \rvert)$
		$(s_9, (pk_j^s, pk_j^r), P_j)$	$(\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$	$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$		$(s_{10}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, pk_j^r, \lvert m \rvert)$
	(s_{10}, pk_j^r, m, P_j)	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Retrieve.Ok}, (pk_j^s, pk_j^r))$	$(\mathcal{S}, \mathcal{S}, \mathbf{PKI.Retrieve.Ok}, P_i, (pk_j^s, pk_j^r))$		$(s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, \lvert m \rvert)$
$(s_6, pk_i^s, pk_i^r, pk_j^r, m, P_j)$	$(s_{11}, -)$		$(P_i, \mathcal{F}_{SC}, \mathbf{SC.Encrypt}, pk_j^r, m)$	$(\mathcal{S}, \mathcal{S}, \mathbf{SC.Encrypt}, P_i, pk_j^r, m)$		$(s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, \lvert m \rvert)$
$(s_9, pk_i^s, pk_i^r, pk_j^r, m, P_j)$			$(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (-, -, pk_j^r), (P_j, pk_j^s, pk_j^r))$	$(\mathcal{S}, B_{SC}^*, \mathbf{B.Lookup}, (-, -, pk_j^r), (P_j, pk_j^s, pk_j^r))$		$(s_{12}, pk_i^s, pk_i^r, P_j, pk_j^r, \lvert m \rvert)$
$(s_{10}, pk_i^s, pk_i^r, pk_j^r, m, P_j)$			$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \mathbf{SC.Encrypt}, pk_i^s, pk_j^r, \lvert m \rvert)$	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \mathbf{SC.Encrypt}, pk_i^s, pk_j^r, \lvert m \rvert)$		$(s_{13}, pk_i^s, pk_i^r, P_j, pk_j^r)$
$(s_{10}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c)$			$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathbf{SC.Ciphertext}, pk_i^s, pk_j^r, c)$	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathbf{SC.Ciphertext}, pk_i^s, pk_j^r, c)$		$(s_{14}, pk_i^s, pk_i^r, P_j, c, pk_j^r)$
$(s_{10}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c)$			$(\mathcal{F}_{SC}, \mathcal{F}_{SC}, \mathbf{SM.Ciphertext}, P_i, P_j, c)$	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Ciphertext}, P_i, P_j, c)$	(s_8, P_j, m, c, pk_j^r)	(s_7, pk_i^s, pk_i^r)
$(s_{10}^{(3)}, pk_i^s, pk_i^r, P_j, pk_j^r, m, c)$			$(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, P_j)$	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, P_j)$	$(s_{10}, P_j, m, c, pk_j^r)$	
$(s_{11}, pk_i^s, pk_i^r, pk_j^r, m, c, P_j)$			$(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Add}, (P_i, P_j, c, m))$	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Add}, (P_i, P_j, c, m))$	$(s_{10}^{(1)}, P_j, pk_j^r, m, c)$	
$(s_{12}, pk_i^s, pk_i^r, c)$			$(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Add}, (pk_i^s, pk_j^r, c, m))$	$(\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B.Add}, (pk_i^s, pk_j^r, c, m))$	$(s_{10}^{(2)}, c)$	
(s_5, pk_i^s, pk_i^r)	(s_{12}, c)		$(\mathcal{F}_{SC}, P_i, \mathbf{SC.Ciphertext}, c)$	$(\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{SC.Ciphertext}, P_i, c)$	(s_{11}, c)	
	$(s_6, -)$		$(P_i, \mathcal{Z}, \mathbf{SM.Ciphertext}, c)$	$(P_i, \mathcal{Z}, \mathbf{SM.Ciphertext}, c)$	$(s_5, -)$	

FIGURE 18. Modified encryption

$M_{SC}(i)$	$M_\pi(i)$	$M_{PKI}(i)$			$M_{SM}(i)$	$M_{\mathcal{S}}(i)$
$(s_0, -)$	$(s_0, -)$	$(s_0, -)$			$(s_0, -)$	$(s_0, -)$
	$(s_{19}, -)$		$(\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$	$(\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$	$(s_{19}, -)$	
	$(s_0, -)$		$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(P_i, \mathcal{Z}, \mathbf{SM.Error})$	$(s_0, -)$	
(s_5, pk_i^s, pk_i^r)	$(s_6, -)$	$(s_5, -)$			$(s_5, -)$	(s_7, pk_i^s, pk_i^r)
	$(s_6^{(2)}, P_j, c)$		$(\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$	$(\mathcal{Z}, P_i, \mathbf{SM.Decrypt}, P_j, c)$	(s_{12}, P_j, c)	
	(s_{13}, P_j, c)		$(P_i, P_i, \mathbf{SM.Decrypt}, P_j, c)$	$(\mathcal{F}_{SM}, \mathcal{S}, \mathbf{SM.Decrypt}, P_i, P_j, c)$	(s_{13}, P_j, c)	$(s_7^{(2)}, pk_i^s, pk_i^r, P_j, c)$
	(s_{14}, c)	(s_6, P_j)	$(P_i, \mathcal{F}_{PKI}, \mathbf{PKI.Retrieve}, P_j)$	$(\mathcal{S}, \mathcal{S}, \mathbf{PKI.Retrieve}, P_i, P_j)$		$(s_{15}, pk_i^s, pk_i^r, P_j, c)$
		(s_7, P_j)	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \mathbf{PKI.Retrieve}, P_i, P_j)$	$(\mathcal{F}_{PKI}, \mathcal{A}_{PKI}, \mathbf{PKI.Retrieve}, P_i, P_j)$		$(s_{16}, pk_i^s, pk_i^r, P_j, c)$
		(s_8, P_j)	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \mathbf{PKI.Retrieve.Ok}, P_i, P_j)$	$(\mathcal{A}_{PKI}, \mathcal{F}_{PKI}, \mathbf{PKI.Retrieve.Ok}, P_i, P_j)$		$(s_{17}, pk_i^s, pk_i^r, P_j, c)$
		$(s_9, (pk_j^s, pk_j^r), P_j)$	$(\mathcal{F}_{PKI}, B_{PKI}, \mathbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$	$(\mathcal{S}, B_{\mathcal{S}}, \mathbf{B.Lookup}, (P_j, -), (P_j, (pk_j^s, pk_j^r)))$		$(s_{17}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, pk_j^r, c)$
	(s_{16}, pk_j^s, c, P_j)	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \mathbf{PKI.Retrieve.Ok}, (pk_j^s, pk_j^r))$	$(\mathcal{S}, \mathcal{S}, \mathbf{PKI.Retrieve.Ok}, P_i, (pk_j^s, pk_j^r))$		$(s_{17}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c)$
$(s_{13}, pk_i^s, pk_i^r, pk_j^s, c, P_j)$	$(s_{17}, -)$		$(P_i, \mathcal{F}_{SC}, \mathbf{SC.Decrypt}, pk_j^s, c)$	$(\mathcal{S}, \mathcal{S}, \mathbf{SC.Decrypt}, P_i, pk_j^s, c)$		$(s_{19}, pk_i^s, pk_i^r, P_j, pk_j^s, c)$
$(s_{14}, pk_i^s, pk_i^r, pk_j^s, c, P_j)$			$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \mathbf{SC.Decrypt}, pk_j^s, pk_i^r, c)$	$(\mathcal{F}_{SC}, \mathcal{A}_{SC}, \mathbf{SC.Decrypt}, pk_j^s, pk_i^r, c)$		$(s_{20}, pk_i^s, pk_i^r, P_j, pk_j^s)$
$(s_{14}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m')$			$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathbf{SC.Plaintext}, pk_j^s, pk_i^r, m')$	$(\mathcal{A}_{SC}, \mathcal{F}_{SC}, \mathbf{SC.Plaintext}, pk_j^s, pk_i^r, m')$		$(s_{21}, pk_i^s, pk_i^r, P_j, m', pk_j^s)$
$(s_{14}^{(2)}, pk_i^s, pk_i^r, P_j, pk_j^s, c, m')$			$(\mathcal{F}_{SC}, \mathcal{F}_{SC}, \mathbf{SM.Plaintext}, P_i, P_j, m')$	$(\mathcal{S}, \mathcal{F}_{SM}, \mathbf{SM.Plaintext}, P_i, P_j, m')$	$(s_{14}, P_j, c, m', pk_j^s)$	(s_7, pk_i^s, pk_i^r)
$(s_{15}, pk_i^s, pk_i^r, pk_j^s, c, m', P_j)$			$(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, P_j, P_j)$	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, P_j, P_j)$	$(s_{14}^{(1)}, P_j, pk_j^s, c, m')$	
$(s_{15}^{(1)}, pk_i^s, pk_i^r, P_j, pk_j^s, c)$			$(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (-, pk_j^s, -), (P_j, pk_j^s, pk_j^r))$	$(\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B.Lookup}, (-, pk_j^s, -), (P_j, pk_j^s, pk_j^r))$	(s_{16}, P_j, c, pk_j^s)	
$(s_{17}, pk_i^s, pk_i^r, pk_j^s, c, P_j)$			$(\mathcal{F}_{SC}, B_{SM}^*, \mathbf{B.Lookup}, (P_j, P_i, c, -), \perp/(P_j, P_i, c, m))$	$(\mathcal{F}_{SM}, B_{SM}, \mathbf{B.Lookup}, (P_j, P_i, c, -), \perp/(P_j, P_i, c, m))$	$(s_{16}^{(1)}, P_j, pk_j^s, c)$	
$(s_{19}, pk_i^s, pk_i^r, m)/(s_{18}, pk_i^s, pk_i^r, \perp)$			$(\mathcal{F}_{SC}, B_{SC}, \mathbf{B.Lookup}, (pk_j^s, pk_i^r, c, -), \perp/(pk_j^s, pk_i^r, c, m))$	$(\mathcal{F}_{SM}, B_{SC}^*, \mathbf{B.Lookup}, (pk_j^s, pk_i^r, c, -), \perp/(pk_j^s, pk_i^r, c, m))$	$(s_{16}^{(2)}, \perp)/(s_{16}^{(3)}, m)$	
(s_5, pk_i^s, pk_i^r)	$(s_{18}, \perp/m)$		$(\mathcal{F}_{SC}, P_i, \mathbf{SC.Plaintext}, \perp/m)$	$(\mathcal{F}_{SM}, \mathcal{F}_{SM}, \mathbf{SC.Plaintext}, P_i, \perp/m)$	$(s_{17}, \perp)/(s_{18}, m)$	
	$(s_6, -)$		$(P_i, \mathcal{Z}, \mathbf{SM.Plaintext}, \perp/m)$	$(P_i, \mathcal{Z}, \mathbf{SM.Plaintext}, \perp/m)$	$(s_5, -)$	

FIGURE 19. Modified decryption

$M_{SC}(i)$	$M_\pi(i)$	$M_{PKI}(i)$			$M_{SM}(i)$	$M_{\mathcal{S}}(i)$
(s_5, pk_i^s, pk_i^r)	(s_8, m)	(s_8, P_j)			(s_7, P_j, m)	$(s_{10}, pk_i^s, pk_i^r, P_j, \|m\|)$
		$(s_8^{(1)}, P_j)$	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B.Lookup}, (P_j, -), (P_j, \perp))$	$(\mathcal{S}, B_{\mathcal{S}}, \textbf{B.Lookup}, (P_j, -), (P_j, \perp))$		$(s_{11}, pk_i^s, pk_i^r, P_j)$
		$(s_8^{(2)}, P_j)$	$(\mathcal{F}_{PKI}, \mathcal{F}_{PKI}, \textbf{SM.Ciphertext}, P_i, P_j, \perp)$	$(\mathcal{S}, \mathcal{F}_{SM}, \textbf{SM.Ciphertext}, P_i, P_j, \perp)$	$(s_8, P_j, m, \perp)$	(s_7, pk_i^s, pk_i^r)
		$(s_{10}, -)$	$(\mathcal{F}_{PKI}, B_{SM}^*, \textbf{B.Lookup}, P_j, \perp)$	$(\mathcal{F}_{SM}, B_{SM}, \textbf{B.Lookup}, P_j, \perp)$	$(s_8^{(1)}, m, \perp)$	
	$(s_9, -)$	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI.Retrieve.Fail})$	$(\mathcal{F}_{SM}, \mathcal{F}_{SM}, \textbf{PKI.Retrieve.Fail}, P_i)$	$(s_9, \perp)$	
	$(s_6, -)$		$(P_i, \mathcal{Z}, \textbf{SM.Not.Registered})$	$(P_i, \mathcal{Z}, \textbf{SM.Not.Registered}, c)$	$(s_5, -)$	
(s_5, pk_i^s, pk_i^r)	(s_{14}, c)	(s_8, P_j)			(s_{13}, P_j, c)	$(s_{17}, pk_i^s, pk_i^r, P_j, c)$
		$(s_8^{(1)}, P_j)$	$(\mathcal{F}_{PKI}, B_{PKI}, \textbf{B.Lookup}, (P_j, -), (P_j, \perp))$	$(\mathcal{S}, B_{\mathcal{S}}, \textbf{B.Lookup}, (P_j, -), (P_j, \perp))$		$(s_{16}, pk_i^s, pk_i^r, P_j)$
		$(s_8^{(2)}, P_j)$	$(\mathcal{F}_{PKI}, \mathcal{F}_{PKI}, \textbf{SM.Plaintext}, P_i, P_j, \perp)$	$(\mathcal{S}, \mathcal{F}_{SM}, \textbf{SM.Plaintext}, P_i, P_j, \perp)$	$(s_{14}, P_j, c, \perp)$	(s_7, pk_i^s, pk_i^r)
		$(s_{10}, -)$	$(\mathcal{F}_{PKI}, B_{SM}^*, \textbf{B.Lookup}, P_j, \perp)$	$(\mathcal{F}_{SM}, B_{SM}, \textbf{B.Lookup}, P_j, \perp)$	$(s_{14}^{(2)}, c, \perp)$	
	$(s_{15}, -)$	$(s_5, -)$	$(\mathcal{F}_{PKI}, P_i, \textbf{PKI.Retrieve.Fail})$	$(\mathcal{F}_{SM}, \mathcal{F}_{SM}, \textbf{PKI.Retrieve.Fail}, P_i)$	$(s_{15}, \perp)$	
	$(s_6, -)$		$(P_i, \mathcal{Z}, \textbf{SM.Not.Registered})$	$(P_i, \mathcal{Z}, \textbf{SM.Not.Registered}, c)$	$(s_5, -)$	

FIGURE 20. Special cases during modified encryption and decryption

[9] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and Systems Sciences*, 28(2):270–299, 1984.

[10] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[11] Birgit Pfitzmann and Michael Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *CCS '00: Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 245–254, New York, NY, USA, 2000. ACM.

[12] Kristian Gjøsteen. Secure Messaging from Signcryption: An Application of Formal Methods to Universal Composability, 2006.