Master Thesis - Spring 2009

Universally Composable Secret Handshakes with Credentials

by

Nathalie Casati

nathalie.casati@a3.epfl.ch

Chauffaud 7
CH-2400 Le Prévoux
Switzerland

Supervised by

IBM Research - Zurich
Jan Camenisch
Thomas Groß
Victor Shoup

EPFL Laboratory For Cryptologic Algorithms
Arjen Lenstra
Martijn Stam

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

IBM Research - Zurich

Abstract

Recently introduced, the *secret handshake* primitive is a new cryptographic protocol that permits two parties to authenticate each other and derive a shared common session key if and only if they belong to the same organization. In this case, both parties learn the affiliation of the other one, otherwise, this information is kept secret.
Authors involved in this field have introduced protocols that include a variety of features from unlinkability between sessions to multi-party secret handshakes. All protocols are proven secure under some cryptographic assumptions, like the Decisional Diffie-Hellman or the RSA assumptions. No protocol has been proven secure in the Universal Composability (UC) framework nor in its extention, the Generalized Universal Composability (UC) framework.
In this thesis, we extend the concept of secret handshakes by presenting a new *secret handshake with credentials* scheme, which allows parties to make a match and derive a shared common session key if and only if they own specific credentials. Camenisch-Lysyanskaya signatures are used to implement credentials and we take advantage of zero-knowledge proofs techniques to make sure that no information about credentials is disclosed.
Our protocol is proven secure in the Generalized Universal Composability (GUC) framework, hence introducing a secret handshake with credentials ideal functionality. It implies that our scheme remains secure under universal composition with other protocols, even if they are run in an adversarial manner.

Acknowledgments

I would like to dedicate this thesis to my parents Chantal and René, who supported me all along my studies. You always showed me love and trust, whatever happened.
I would like to thank my advisor Thomas Groß, Jan Camenisch and Victor Shoup, who have taken a lot of time to talk with me and transmit their knowledge. I really appreciated those moments.
I would like to specially thank Martijn Stam and my professor Arjen Lenstra for reviewing this thesis and pointing out useful ideas and comments.
Last but not least, I would like to thank my friends at the Zurich Research Lab. You have made my time here so much more enjoyable. I will always remember all of our experiences, especially those organized by the hobby club.

Contents

1 Introduction

An important part of today's transactions are performed through the Internet. As the need for information privacy is growing, confidence in provided data can be significantly raised if a certain trust level can be achieved regarding attributes provided by parties. For instance, during such a transaction, one of them proves having the required amount of money and the second one, the possession of the sought object. A set of these attributes, which include private information, signed by an issuer and bound to some party is what we call a *credential*. A *secret handshake with credentials* protocol is an evolution of the secret handshake cryptographic primitive. It allows two parties to make a match and derive a shared common session key if and only if they own specific *credentials*. Otherwise, if they don't, parties cannot make any conclusion about the veracity of those credentials. As another example, one can also prove being over eighteen by showing one's identity card or having the right to drive a car by showing the appropriate driver's license. However, from a privacy point of view, one does not want to show one's ID card or driver's license because it would reveal name and one's birthdate. In order to hide these details, we use zero-knowledge proofs, which allow proving that one owns a credential without revealing anything about it.

Original definition of secret handshakes. Introduced in 2003 [BDS+03], the secret handshake primitive does not have a notion of credential. Instead, the two parties derive a common session key if and only if both of them are part of the same *group* (i.e., the same organization of people), otherwise none of them learns the group affiliation of the other one. In this case, the credential is limited to group membership, while our scheme permits the use of generic credentials, as explained above.
A secret handshake scheme is usually composed of four different algorithms:

Setup is a trusted algorithm that, given a security parameter, outputs public parameters common to all subsequently generated groups.

CreateGroup must be run by a group administrator and permits to create a new group, given the public parameters. It outputs the group secret key and the group public parameters.

AddMember must also be run by a group administrator and permits to add a new user to a group, given the group public parameters, its secret key and the public parameters. It outputs the membership credentials of the new user.

Handshake is a protocol run between two parties A and B, given the public parameters and private group memberships of A and B. It ensures that a match is achieved if A and B are part of the same group. In this case, they learn each other's group affiliation. Otherwise, neither A or B learns anything about the group affiliation of the other party. Moreover, a third party observing the exchange should not be able to conclude anything about the membership of A and B, nor about the specific group identities.

Secret handshake protocols have well evolved; while the original protocol of [BDS+03] is a two-party protocol allowing usage of roles within a group, state of the art protocols allow a diversity of additional features, which include:

- unlinkability between sessions
- multi-party secret handshakes
- multi-group secret handshakes (the match is achieved if parties are part of (exactly) the same groups)
- reusable credentials (the group membership "ticket" can be reused without losing unlinkability)
- revocation
- dynamic matching (each party can specify both the group and the role the other must have in order to complete the handshake)
- fuzzy matching (allows a match if at least a number of attributes are the same)

The latter property is similar to credential-based matching, but nevertheless different; as long as parties have a number of attributes in common (less than all provided attributes), it is enough to achieve a match. All previous secret handshake schemes base their security on standard cryptographic assumptions like Computational or Bilinear Diffie-Hellman and RSA, and some of them are proven secure in the random oracle model (ROM) [BR93]. Moreover, they only provide group matching (with the exception of [ABK07] which provides attribute-matching using the fuzzy matching technique).
Secret handshake protocols must satisfy two main security properties, as defined in [BDS+03]. The first one, called *impersonator resistance* is violated if an honest party V, who is a member of group G, authenticates an adversary A as a group member, even though A is not a member of G. The second one, referred to as *detector resistance*, is violated if an adversary A can decide whether some honest party V is a member of some group G, even though A is not a member if G. Those two properties are usually formulated as security games. For completeness, we include them in Appendix A.
In order to achieve authentication, secret handshake schemes are designed using two different paradigms. Either a key agreement protocol is run where keys are equal if and only if parties are in the same group. In this case the protocol outputs a key. Or parties exchange encrypted values that can be recovered if and only if both parties are in the same group. In addition, parties have to provide a proof that the values have been decrypted. The protocol then outputs accept or reject. Our protocol uses the first paradigm, and outputs a shared common session key if each party can convince the other that it owns a specific credential.
Several orthogonal problems come to mind when considering the definition of secret handshakes (with credentials). Firstly, parties having performed a successful secret handshake will continue to communicate afterwards, while parties who failed to authenticate each other will not. Previous works do not address this issue, but recommend using anonymous channels or steganographic techniques [XY04, TX06]. Another problem is that, by definition, secret handshake schemes are unfair, in the sense that there is always one party learning the group affiliation of the other one first and could decide to abort the protocol at this point. As previous works does, we consider these issues as orthogonal to our work.

Our secret handshake with credentials protocol. Our new secret handshake protocol is different from former secret handshake schemes, in the sense that groups as such do not really exist, but are implicitly defined by people owning a specific credential. For instance the "group" of car owners consists of the people possessing a car ownership credential. Consequently, our goal is not to authenticate two users of the same group, but to authenticate two parties owning specific credentials. As our protocol uses credentials instead of group memberships, parties

have to agree on these credentials before the start of the protocol execution. To achieve this, parties have to perform a *policy negotiation* step. This consists in exchanging the nonsecret part of their credential over an encrypted channel. In the remainder of the document, we call this part of the credential, the *statement*. Former secret handshake protocols can avoid the policy negotiation step, because they always use their group membership as a credential. Let us consider using our protocol with group memberships as credentials. If there is only one group in the system, the adversary can trivially guess the group identity, which contradicts the goal of the Handshake algorithm, as cited above. However, if there is more than one group in the system, parties would leak information to each other during the policy negotiation step, while agreeing on which group they choose to perform the handshake. Hence, we limit our protocol to policies that are unrelated to each other.
We implement Credentials themselves using Camenisch-Lysyanskaya signatures [CL02]. This allows proving relations between credentials, like combining several of them using AND and OR *zero-knowledge* proofs, proving equality of attributes, or that attributes lie in a given interval. With this technique, one can prove that a party owns a credential without disclosing any secret information about it. For instance, one can prove that a party has the right to drive a car without actually showing its driver's license to anybody and that this party is over eighteen without showing its ID card.
Furthermore, we highlight that credential issuance is not modeled as a part of the system; credentials are issued by accredited authorities and given to parties prior to running the protocol. As a result of the difference between our scheme and former secret handshake protocols, we stress that the two main security properties of former secret handshake schemes, as defined in the previous paragraph, are not relevant anymore. In effect, they strongly imply a notion of group, which we don't have. Nevertheless, we informally show a way of satisfying a modified version of them, similar to what is done in [ABK07].

Practical Applications. Secret handshake protocols have similarities with password-authenticated key agreement schemes [BM92], where two parties derive a common session key if and only if they provide the same password during the protocol execution. However, this approach implies that these parties already know of each other, since they had to agree on the password used during an earlier phase.
Our secret handshake with credential protocol open doors to solve a new privacy and security issue that has not been addressed before. In the PAKE setting, the issuer of the "passwords" or credentials becomes a verifier at a later stage, for when a party wants to authenticate itself and open a secure communication channel. Our protocol makes it easier to separate those roles, as the issuance of credentials was performed as an earlier stage by an accredited issuer. Our protocol lets parties play the role of mutual verifiers for their credentials.
Practical examples include authentication on internet sites where privacy is important. For instance, most internet forums require the user to be above thirteen in order to post messages, however, it is sufficient to click the *yes* button, even if one does not fulfill the age limit. In order to really prove this, one could send a copy of one's ID card (credential issued by the state where the person lives) to the owner of the forum. But this is too slow and is not an option for privacy concerns: one does not want to reveal other private details written on the card. Moreover, nothing proves that the ID card is original. Instead, the user can perform a secret handshake with a credential stating that she is over thirteen. The server's credential would be its accreditation to receive such age information and a certificate that it really runs the forum. This part of the proof is important to make sure that private information is only sent to authorized

parties, otherwise users could be victims of phishing or other impersonation techniques.
We can also imagine a TV channel with limited budget, which streams its videos through a video streaming service on the Internet. On the one hand, the user has to prove that she has paid her subscription fees in order to watch the videos, and that she is above a predefined age limit. On the other hand, the server running those websites has to prove that it is accredited to get age information and substract subscription points from the user's card. This method allows video streaming providers to not share private information about users, such as forcing them to share accounts with the TV channel organization or, inversely, telling the issuer of the credential who they are. Moreover, with this technique, users do not have to log in using a password, but can use credentials instead.
However, in these two examples, the server does not have anything private to hide to unregistered users, which leads us to the following example: the realization of a private online dating website. Let us imagine two users, Alice and Bob, who want to meet using the dating application. The website provides users with a list of details to enter in their profile. In this example, we use the height, the eye colour, the sex and the birthdate. The four of them are certified attributes and appear on Alice's and Bob's identity card, which they can use as a credential. During the policy agreement phase, Alice specifies that she is looking for a boy with blue or green eyes, taller than 1m78 and between 25 and 30 years old, while Bob is looking for a girl with brown eyes and younger than 30. Next, they perform a secret handshake with credentials and if the criteria Bob or Alice provide do not exactly match the other's policy, none of them learn anything about the other's attributes. However, if the policies match the credentials, both of them will learn that the provided details matched their requirement, and they will derive a shared common session key that will allow them to start a private communication. We can imagine that some details are publicly available in their user profiles, in order to first select someone to perform a handwith with. Moreover, if the users' credentials are not stored on the server, but on their own machine, the server running the dating website will not be able to guess if users have achieved a match or not.
Such applications are not possible using former secret handshake schemes because they only allow parties to provide credentials that have a strong relation between each other (the same group), while, here, we use the fact that they are unrelated to each other.

Our contribution. Our new secret handshake with credentials protocol allows to authenticate parties based on credentials, instead of group membership. This is implemented using Camenisch-Lysyanskaya signatures and zero-knowledge proofs.
Moreover, it is the first secret handshake protocol to be proven secure in the Generalized Universal Composability (GUC) framework [CDPW07], which is an extension of the Universal Composability (UC) framework of [Can01]. Both provide a new way of proving security for protocols, using *ideal functionalities*. Hence, our ideal functionality for secret handshake with credentials "emulates" our protocol, so that, only attacks that can be performed on the ideal functionality can be applied to our protocol. Of course, the ideal functionality is designed in such a way that it is not vulnerable to any attacks. Using the GUC framework, security is maintained under composition with other secure protocols in the same model, even when running concurrently with any number of arbitrary protocols controlled by the adversary. This allows a modular design of more complex schemes, by splitting them into simpler sub-protocols.

Outline of this thesis. This work is organized as follows. In Section 2, a brief overview of the related work is given. In Sections 3 and 4, necessary definitions and preliminary background

are reviewed. In Section 5, the service interface of the secret handshake with credentials ideal functionality is presented and in Section 6, the corresponding ideal functionality is introduced. In Section 7, the needed setup assumptions are exposed. In Section 8, the zero-knowledge ideal functionality is reviewed. In Section 9, we present our new protocol and prove its security in Section 10. In Section 11, we show how our protocol can use Camenisch-Lysyanskaya signatures as credentials. Finally, we conclude with future work suggestions in Section 12.

2 Related Work

Since 2003, about fifteen papers have been written on secret handshakes. The most important ones will be cited here. The original paper by Balfanz et al. [BDS+03] introduces the primitive itself as a two-party protocol together with a notion of roles, and implements it using pairing-based cryptography. The proposed solution is simple but requires the use of one-time credentials to achieve unlinkability.
Castelluccia, Jarecki and Tsudik presented a protocol [CJT04] that, while being slightly more efficient, still doesn't allow reusable credentials . It is based on CA-oblivious encryption and is secure in the Random Oracle Model (ROM).
Xu and Yung provided a solution [XY04] which doesn't need ROM and supports reusable credentials at the cost of anonymity; in the worst case, a participant can be identified as one out of k people by an adversary.
[TX06, JKT06, JKT07] are all multi-party versions of the secret handshake primitive with different properties.
Ateniese and Blanton [ABK07] introduced the notions of dynamic and fuzzy matching by presenting new protocols which do not rely on ROM, permit reusable credentials and unlinkability. Dynamic matching was already used in [CJT04] and means that a participant can specify the group (and role) that the other user must have in order to complete the handshake. Fuzzy matching approaches our notion of secret handshakes with credentials; the handshake is successful if, for each user, a parameterizable number of credentials d out of the total number of credentials is matching at least d credentials of the other player, and vice versa. While allowing usage of policies that relate to each other, this technique does not provide certitude regarding which credentials are really possessed by the parties.
Jarecki and Liu [JL07] presented the first unlinkable scheme that provides practical revocation features, while strengthening definitions of secret handshakes schemes.
The protocol of [JKT08] guarantees security under arbitrary protocol composition and is secure under the RSA assumption, using the random oracle model.
[YT07, KTK+08] presented secret handshale schemes that allow two users to authenticate each other if and only if they are part of (exactly) the same groups. These schemes are referred to as secret handshakes with multiple groups.
Throughout this work, we do not concentrate on providing features that have been implemented by previous schemes, but focus on designing a new protocol that makes use of credentials instead of group membership, and that is universally composable, something that has not yet been done by authors previously involved in this field.

3 Definitions

In this section, the definitions and notations needed throughout this work are introduced.

3.1 Cryptographic Assumptions

In cryptography, problems believed to be hard to solve are often used to computationally prove the security of schemes or protocols. With hard, we mean that no probabilistic polynomial time (PPT) algorithm can solve the problem with non-negligible probability. Using reduction techniques, the proof outlines that if the adversary can break the security of the analyzed protocol, then it can also break the considered problem. However, as the latter is believed to be hard, a computationally limited adversary should not be able to solve it. This shows that the protocol has the required level of security. These problems are also referred as assumptions.

3.1.1 The Decisional Diffie-Hellman Problem

The Decisional Diffie-Hellman problem is defined as follows. Given $T = g^t, D = g^d, R = g^r \in \mathbb{G}$, output 1 if $td = r \bmod q$, 0 otherwise. An adversary can solve the DDH problem if it can output the correct answer with non-negligible probability.

3.1.2 The Decisional Diffie-Hellman Assumption

Let $\mathbb{G}$ be a cyclic group in which the Decisional Diffie-Hellman problem (as defined above) is hard. Let q be the order of $\mathbb{G}$, where q is a large prime. Let g be a generator of $\mathbb{G}$ and let t, d, r be three random elements of $\mathbb{Z}_q$. The Decisional Diffie-Hellman assumption states that the distribution (g, g^t, g^d, g^{td}) is computationally indistinguishable from the distribution (g, g^t, g^d, g^r).
The DDH assumption and the DDH problem are equivalent, and there exist other equivalent formulations [CS98].

3.1.3 The Strong RSA Assumption

The strong RSA assumption states that it is computationally infeasible, on input of a random RSA modulus n and a random element $u \in \mathbb{Z}_n^*$, to compute values $e > 1$ and v such that $v^e \equiv u \pmod{n}$.

3.2 Message Authentication Codes

A MAC or Message Authentication Code is a value computed by the sender from a key and a message in order to authenticate the latter. Using the same key and the message, the receiver can recompute the MAC value and check it against the one he received from the sender. If the message has not been altered, the MAC values are identical. Otherwise they differ.
More formally, a MAC scheme is composed of a function $\mathrm{MAC}_K(m)$ to compute a MAC value from the key K and the message m. It maps an input of finite length to an output of fixed

length. The key used has to be unknown to the adversary. In order to be secure against existential unforgeability under chosen-message attacks (EUF-CMA) [GMR88], a MAC scheme should respect the following property. Let $\mathcal{O}$ be a MAC generation oracle which outputs MAC values $\text{MAC}_K(m_i)$ on input messages m_i. Given access to $\mathcal{O}$, it is computationally infeasible to compute any valid new pair $(m, \text{MAC}_K(m))$, where $m \neq m_i$.

3.3 Key Derivation Functions

A Key Derivation Function is a function that takes as input a secret value (in our case it will be a group element) and outputs a key of fixed length. It uses a deterministic algorithm. Its aim is to hide the secret value from an adversary that may find a key derived in this way. We will refer to this type of function as $\text{KDF}(\cdot)$.

4 Preliminaries

4.1 Universal Composability Framework

Introduced by Canetti in 2001, the Universal Composability (UC) framework [Can01] provides a new way of proving security for protocols. Security is maintained under composition with other secure protocols in the same model, even when running concurrently with any number of arbitrary protocols controlled by the adversary. This allows a modular design of more complex schemes, by splitting them into simpler sub-protocols. The idea is to prove that a protocol is at least as secure as an ideal process for the same goal.
A variety of cryptographic primitives have already been studied in the UC framework. These include digital signatures, public-key encryption, commitment, zero-knowledge, key exchange and password authenticated key-exchange. However secret handshake protocols have not yet been studied in the UC framework.
In the remainder of the section, the knowledge about the Universal Composability framework that is necessary to understand this document is explained. First we present the intuition of the model, how to model parties and how they communicate. Next we describe the execution of a real-world protocol in the presence of a real-world adversary and formalize the concepts of ideal process and ideal functionalities. We define what it means for a protocol to securely realize an ideal functionality for the same goal. We present the hybrid model and the composition theorem. Finally, we introduce the Generalized Universal Composability framework (GUC).

Intuition of the model. In order to determine whether a protocol is secure for a specific cryptographic goal, we make use of an *ideal process* that models the same the task in secure way. In the ideal process, parties give their inputs to a trusted third party (referred to as an *ideal functionality*) which computes the outputs locally and hands them back to the parties. The ideal process can be viewed as a formal specification of the studied protocol: which properties it should have, which imperfections, etc. A protocol is said to *securely realize* a specific task if an execution the of the former "emulates" the ideal process for the task, that is any successful attack performed by the adversary on the while interacting with the protocol is also possible in the ideal process. However as the latter is secure, this is not possible. This demonstrates the security of the protocol.

Modeling parties. Each entity is modeled as an Interactive Turing Machine (ITM) [GMR89], and represents the code of a specific party, which can be honest or adversarial. Each ITM can be instantiated several times: we call one of these instances an ITM Instance (ITI). One can view an ITM as a piece of code that can be run multiple times to form multiple instances of the same code. These ITMs are interconnected through a network, represented by their shared tapes on which ITMs can read or write. An execution of the system corresponds to a series of activations where one single ITI is active at any given time. This ITI may write to the input tape of only one other ITI, after which it enters a special waiting state. The ITI to which it wrote becomes active. The execution always starts with a special ITI that is active called the environment. It is referred to as $\mathcal{Z}$ throughout this thesis. It can be viewed as the entity starting the process we are studying. It starts its execution on an external input and is activated if an ITI does not write to any tape. The output of the overall execution or system is the final output of $\mathcal{Z}$.
In order to identify a set of ITIs running together to form a protocol instance (i.e., a protocol session), these ITIs will receive the same Session Identifier (SID). Each ITI also has its proper Party Identifier (PID) to describe its role within the execution. The identity of an ITI, that is, the tuple (PID, SID) is unique in the system.

Protocol execution in the real-life model or real world. Special ITIs are the environment $\mathcal{Z}$ and the real-world adversary $\mathcal{A}$. The execution of the real-world protocol, which is referred to as Π throughout this section, is unauthenticated and asynchronous. In order to match real life best, the network is modeled as unreliable, and the adversary can observe and delay messages. Once the latter has been activated, it can deliver a message to some party or to $\mathcal{Z}$, or corrupt a party, so that it can access that party's internal state and control its actions. $\mathcal{Z}$ is notified when corruptions occur. The adversary $\mathcal{A}$ has no access to inputs or outputs of noncorrupt parties, only to their communication over the network.
At the beginning of the protocol execution, $\mathcal{Z}$ is activated and then invokes the adversary $\mathcal{A}$. The next activations follow the rules explained above. Within the execution, each entity has the same SID and parties get their protocol inputs from $\mathcal{Z}$.

Ideal process and ideal functionalities. Special ITIs are the environment $\mathcal{Z}$, an ideal functionality $\mathcal{F}$ and the ideal-world adversary $\mathcal{S}$. The latter is often viewed as a *simulator* because in the proof of security $\mathcal{S}$ will have to simulate the real-world adversary $\mathcal{A}$. In the ideal world, parties are called *dummy* parties because all they do is forward messages they receive from $\mathcal{Z}$ to the ideal functionality $\mathcal{F}$ and inversely. Hence, in order to corrupt a party, $\mathcal{S}$ sends a *corrupt* query to $\mathcal{F}$. The answer to that query is the information $\mathcal{S}$ gets after having corrupted the party. After having corrupted a party, $\mathcal{S}$ controls the party's actions. $\mathcal{Z}$ and of course also $\mathcal{F}$ are notified when corruptions occur. The former, which can be seen as a trusted third party, has the same functionality as protocol Π and is said to be ideal because it is designed in such a way that nothing can go wrong. Hence it replaces the protocol by a perfect execution called an ideal process. Dummy parties and $\mathcal{S}$ may provide inputs to $\mathcal{F}$, which then generates outputs according to them, just like a predefined function would. $\mathcal{S}$ has the possibility to delay and observe these outputs. An observed and delayed output is referred to as *public delayed*, or if it is only delayed, it is referred to as *private delayed* in the definition of our ideal functionalities. $\mathcal{F}$ also keeps an internal state. The execution of the ideal process follows the same rules as in the real-world model. Within this execution, $\mathcal{F}$ differentiates parties with their PIDs. As usual, all participating ITIs share the same SID.

Securely realizing an ideal functionality. A protocol Π securely realizes an ideal functionality $\mathcal{F}$ if Π successfully emulates the ideal process for $\mathcal{F}$. In other words, if the environment $\mathcal{Z}$ is unable to distinguish between the output of an execution of Π and $\mathcal{A}$ in the real world or between the one from an execution of $\mathcal{F}$ and $\mathcal{S}$ in the ideal world, then Π is at least as secure as $\mathcal{F}$. Moreover, any information that $\mathcal{A}$ can obtain during a run of Π in the real world can also be obtained by $\mathcal{S}$ in the ideal world, with access only to $\mathcal{F}$.
When a protocol Π securely realizes an ideal functionality $\mathcal{F}$ in the UC framework, we say that protocol Π UC-realizes the ideal functionality $\mathcal{F}$. The common theorem outline states that a protocol Π securely realizes an ideal functionality $\mathcal{F}$ if for any real-word adversary $\mathcal{A}$ there exists an ideal-world adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can distinguish whether it is interacting with A and parties running Π in the real world or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal world. This means that in order to prove this, we have to build the simulator $\mathcal{S}$ and show that the simulation is successful.
During the simulation, simulator $\mathcal{S}$ plays the role of a corrupt party B^* and runs the ideal protocol together with an honest party A. To be able to hand in a proper input to $\mathcal{F}$, $\mathcal{S}$ has to simulate a run of the real-world protocol Π. In order to do so, $\mathcal{S}$ has access to a copy of the ideal-world adversary $\mathcal{A}$. All messages from $\mathcal{Z}$ addressed to $\mathcal{S}$ are directly forwarded to $\mathcal{A}$ and all outputs of $\mathcal{A}$ are given back to $\mathcal{Z}$. Throughout the real-world protocol simulation, $\mathcal{S}$ plays the role of A and $\mathcal{A}$ the role of B^*. The goal of the simulator is to extract B^*'s input, which was given to him directly by $\mathcal{Z}$. As input to protocol Pi, $\mathcal{S}$ uses information leaked by the ideal A during the run of the ideal process. The situation is represented in Figure 1.

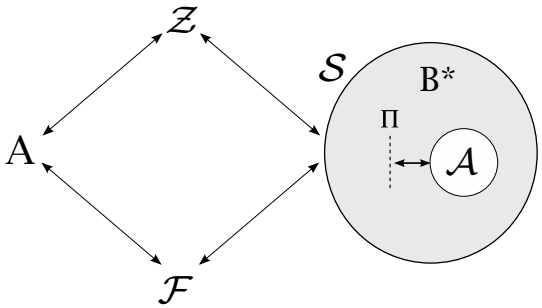
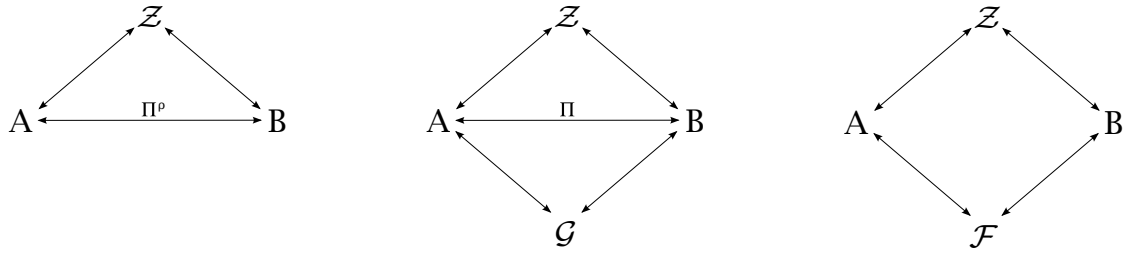


Figure 1: The simulator $\mathcal{S}$

The hybrid model. The hybrid model is similar to the real-word model, except that protocol Π has access to an unbounded number of instances of ideal functionalities. Let us formulate the $\mathcal{G}$-*hybrid* model, where Π can make calls to instances of $\mathcal{G}$.
The protocol execution happens in the same way as in the real-word model. In addition, each copy of $\mathcal{G}$ is identified using a different SID, which allows Π to differentiate them. When a message is sent to a particular copy of $\mathcal{G}$, this copy is activated next. If it does not yet exist, it will be first created. Note that the SIDs are chosen by protocol Π.
The model is interesting because it allows a modular design of secure protocols, which works as follows. First one designs Π, which securely realizes the ideal functionality $\mathcal{F}$ in the $\mathcal{G}$-*hybrid* model. Second, one designs ρ, which securely realizes $\mathcal{G}$. Finally one replaces $\mathcal{G}$ by ρ in the execution of Π: each call to $\mathcal{G}$ is replaced by a call to a corresponding instance of ρ using the same SID with the same input. This newly composed protocol is referred to as Π^ρ, which has the same security as Π run in the $\mathcal{G}$-*hybrid* model, as stated by the theorem given in the next

paragraph.
On Figure 2, you can see a graphical representation of the real, hybrid and ideal world. Parties are called A and B. Note that in the ideal world $\mathcal{G}$ does not exist but is simulated by $\mathcal{S}$.



(a) Real World (b) Hybrid World (c) Ideal World

Figure 2: Real, hybrid and ideal world

The composition theorem. In its general form, the composition theorem states that if ρ securely realizes $\mathcal{G}$ in the $\mathcal{H}$-*hybrid* model and Π securely realizes $\mathcal{F}$ in the $\mathcal{G}$-*hybrid* model, then for any adversary $\mathcal{A}_{\mathcal{H}}$ there exists an adversary $\mathcal{A}_{\mathcal{G}}$ such that no $\mathcal{Z}$ can distinguish a run of the composed protocol Π^{ρ} and $\mathcal{A}_{\mathcal{H}}$ in the $\mathcal{H}$-*hybrid* from a run of Π and $\mathcal{A}_{\mathcal{G}}$ in the $\mathcal{G}$-*hybrid* model in any way.
An immediate corollary of this theorem says that if ρ securely realizes $\mathcal{G}$ in the $\mathcal{H}$-*hybrid* model and Π securely realizes $\mathcal{F}$ in the $\mathcal{G}$-*hybrid* model, then Π^{ρ} securely realizes $\mathcal{F}$ in the $\mathcal{H}$-*hybrid* model. A second corollary states that if ρ securely realizes $\mathcal{G}$ in the real-life model and Π securely realizes $\mathcal{F}$ in the $\mathcal{G}$-*hybrid* model, then Π^{ρ} securely realizes $\mathcal{F}$ in the real-life model.

The Generalized Universal Composability framework. Often, cryptographic protocols are designed in such a way that they need to have access to some common global setup information and this is also the case for ours. However, the UC framework fails to guarantee composable security if protocols share trusted setup information. To solve this problem, the Generalized Universal Composability (GUC) framework was introduced by Canetti et al. in 2007 [CDPW07]. Informally, the new notion of security states that a protocol Π that securely realizes an ideal functionality $\mathcal{F}$ in the GUC framework using some global setup does not affect the security of any other protocols more than $\mathcal{F}$ does, even if protocols running concurrently with Π are maliciously constructed and all protocols use the same global setup.
In the GUC framework, setup is modeled with the help of an additional trusted entity, called a shared functionality, which is an ITI with the same properties as an ideal functionality except that it may communicate with parties that are not running the protocol, such as the environment $\mathcal{Z}$, or any other entity even if it runs another session of the same or another protocol. This also means that a shared functionality exists in both the ideal and the real world.
A major difference between the UC and the GUC framework is the constraints the environment $\mathcal{Z}$ is subject to when performing the distinguishing experiment. In the UC framework, $\mathcal{Z}$ must be able to choose the challenge protocol inputs, observe the latter's outputs and communicate with the attacker, which may either be $\mathcal{A}$, the real-world adversary, or $\mathcal{S}$, the ideal-world adversary. Constraining the environment to these possibilities only and only allowing it to interact with a single session of the challenge protocol is sufficient to achieve the security goal of the composition theorem as described above, whereas the environment in the GUC framework is

unconstrained. It means that during the distinguishing experiment, $\mathcal{Z}$ can interact with an arbitrary number of protocols, including multiple sessions of the challenge protocol. Some of these protocol sessions may share state with the challenge protocol sessions. The only contraint on $\mathcal{Z}$ is that it is not allowed to observe or influence the network communications of the challenge protocol sessions, but this is the job of the adversary (which is controlled by $\mathcal{Z}$). Hence, the GUC distinguishing experiment allows a very powerful setting, but is also a lot more complex to master, which significantly increases the difficulty of proving the security of protocols in the GUC framework.
In order to simplify this process, the notion of Externalized UC (EUC) framework is introduced. In practice, protocols share state information in a very restricted way, which can be modeled by letting them access only one shared functionality. The latter is referred to as $\bar{\mathcal{G}}$ (the bar is added to differentiate shared from ideal functionalities) and its SID is required to begin with the special symbol #, which is exclusively used in the SID of shared functionalities. Such a protocol that only share state via a single shared functionality $\bar{\mathcal{G}}$ is called a $\bar{\mathcal{G}}$-subroutine respecting protocol. Considering only $\bar{\mathcal{G}}$-subroutine respecting protocols makes the task of proving security simpler because this allows one to constrain the environment $\mathcal{Z}$ again. In addition to being subject to the same constraints as in the basic UC model, $\mathcal{Z}$ is allowed to invoke a single external protocol, namely the protocol for the shared functionality $\bar{\mathcal{G}}$. Such an environment is called a $\bar{\mathcal{G}}$-externally constrained environment.
When a subroutine respecting protocol Π securely realizes an ideal functionality $\mathcal{F}$ invoking no other shared functionalities than (possibly) $\bar{\mathcal{G}}$ in the GUC framework, we say that protocol Π GUC-EUC-realizes the ideal functionality $\mathcal{F}$.
Throughout this work the use of $\bar{\mathcal{G}}$-subroutine respecting protocols is sufficient, so we limit ourselves to EUC security. However it has been shown in [CDPW07] that security in the EUC framework is strictly equivalent to security in the GUC framework.

4.2 The Camenisch-Lysyanskaya signature scheme

Camenisch and Lysyanskaya introduced CL signatures in 2002 [CL02]. They show efficient protocols around the scheme that are useful for the creation of a credential system. Specifically, it is possible to prove ownership of a credential while disclosing nothing about the signature itself. This is described in Section 4.3.
The signature scheme allows one to sign messages $m = (m_0, ..., m_{L-1})$ such that $m_i \in \pm\{0,1\}^{l_m}$. It is described in Figure 3. For more details, see [CL02].
The CL signature scheme is secure against adaptive chosen messages attacks [GMR88] under the strong RSA assumption as defined in Section 3.1.3.

4.3 Zero-Knowledge Proofs

The concept of zero-knowledge proofs aims at solving a specific problem: how does one prove the possesion of information to somebody without transferring this information, that is, without disclosing it? Zero-knowledge proofs have been conceived in 1989 by Goldwasser, Micali and Rackoff [GMR89], who presented the first concrete example of such a proof: showing that a number is quadratic nonresidue mod m, releasing no additional knowledge. It has been proven that all languages in NP have zero-knowledge proofs [GMW91].

Key generation:
On input l_n, choose an l_n-bit RSA modulus $n = pq$ such that p and q are safe primes. Choose uniformly at random $R_0, \ldots, R_{L-1}, S$ and $Z \in \mathrm{QR}_n$. Output $(n, R_0, \ldots, R_{L-1}, S, Z)$ as the public key and p as the secret key.
Signing a message $m = (m_0, ..., m_{L-1})$**:**
Choose a random prime e of length $l_e > l_m + 2$ and a random number v of length $l_v = l_n + l_m + l_r$, where l_r is a security parameter. The signature on the message $(m_0, ..., m_{L-1})$ is (A, e, v), where

$$A = \left(\frac{Z}{R_0^{m_0} \cdots R_{L-1}^{m_{L-1}} S^v} \right)^{1/e} \mod n.$$

Verifying a signature (A, e, v)**:**
Check that $Z \equiv A^e R_0^{m_0} \cdots R_{L-1}^{m_{L-1}} S^v \pmod{n}$, $m_i \in \pm\{0,1\}^{l_m}$ and $2^{l_e} > e > 2^{l_e - 1}$.

Figure 3: The Camenisch-Lysyanskaya signature scheme.

More formally, a zero-knowledge proof is a two-party protocol, involving a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. They received a statement x and a relation R as common input. In addition, prover $\mathcal{P}$ receives the witness w as extra input. The task of prover $\mathcal{P}$ is to convince verifier $\mathcal{V}$ that he owns a witness w such that $(x, w) \in R$. If so, $\mathcal{V}$ should accept. If $\mathcal{P}$ doesn't have a w such that $(x, w) \in R$, $\mathcal{V}$ should reject. Moreover the protocol should be zero-knowledge in the sense that it should leak no information about the witness w except that there (possibly) exists a w such that $(x, w) \in R$.
An interactive proof protocol is said to be a zero-knowledge proof of knowledge if it has the following three properties:

Completeness Given $(x, w) \in R$, the protocol is complete if it succeeds with overwhelming probability (i.e., the honest verifier $\mathcal{V}$ verifies the proof successfully), when it is run by an honest prover $\mathcal{P}$. The definition of overwhelming depends on the concept of the application, but generally implies that the probability of failure is not of practical significance.

Soundness Given $(x, w) \notin R$, the protocol is sound if there exists an expected polynomial-time algorithm M with the following property: if a dishonest prover (impersonating $\mathcal{P}$) can with non-negligible probability successfully execute the protocol with an honest verifier $\mathcal{V}$, then M can be used to extract from this prover knowledge (essentially equivalent to $\mathcal{P}$'s secret) which with overwhelming probability allows successful subsequent protocol executions.

Zero-knowledge Given $(x, w) \in R$, the protocol is zero-knowledge if it is simulatable in the following sense: there exists an expected polynomial-time algorithm (*simulator*) which can produce, upon input of the assertion(s) to be proven but without interacting with the real prover $\mathcal{P}$, transcripts indistinguishable from those resulting from interaction with the real prover $\mathcal{P}$.

Nowadays, zero-knowledge proofs can also be used to efficiently prove a variety of different relations between secret and public values:

- AND-proofs
- OR-proofs
- proofs of equality of discrete logarithms
- proofs that a value lies in a given interval

We review the Camenisch-Stadler notation, which serves the purpose of expressing such proofs.

The Camenisch-Stadler notation. Introduced in [CS97], the Camenisch-Stadler notation permits to show the goal of a zero-knowledge proof in a very succinct way, while hiding the details of the proof. The following equation shows an example of this notation.

$$PK\{(\alpha, \beta, \gamma) : y = g^{\alpha} h^{\beta} \wedge \tilde{y} = \tilde{g}^{\alpha} \tilde{h}^{\gamma} \wedge (v < \alpha < u)\}$$

It denotes a zero-knowledge proof of knowledge of integers α, β and γ such that $y = g^{\alpha} h^{\beta}$ and $\tilde{y} = \tilde{g}^{\alpha} \tilde{h}^{\gamma}$ holds, with $(v < \alpha < u)$, where $y, g, h, \tilde{y}, \tilde{g}$ and $\tilde{h}$ are elements of some groups $G = \langle g \rangle = \langle h \rangle$ and $\tilde{G} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$. The values in parenthesis are the secret quantities of knowledge and the relation to be proven is stated after the colon. By convention, the secret values, i.e., what is being proven, are represented by greek letters, and normal letters are public parameters used in relation with these secrets.
An example of an OR-proof is shown in the following equation.

$$PK\{(\sigma_1, \sigma_2) : h_1 = g^{\sigma_1} \vee h_2 = g^{\sigma_2}\}$$

The four types of proofs cited above are called zero-knowledge proofs of representation.
Another type of zero-knowledge proof will be referred to in this document: proofs of possession of a signature, namely a CL signature [CL02, Cam06]. The aim of such proofs is to convince a verifier $\mathcal{V}$ that one possesses a signature on some messages. In our case, we consider that $\mathcal{V}$ is not privy to these messages. We want to build a zero-knowledge proof showing that the signature (A, e, v) is valid, i.e. that $Z \equiv A^e R_0^{m_0} \cdots R_{L-1}^{m_{L-1}} S^v \pmod{n}$, $m_i \in \pm\{0,1\}^{l_m}$ and $2^{l_e} > e > 2^{l_e - 1}$. If A was a public value, then we could build a zero-knowledge proof of possession of a CL signature by combining zero-knowledge proofs of representation as seen above. However, making A public would leak information about the signature, which is exactly what we want to prevent using the zero-knowledge proof technique. A solution to this is to randomize A in such a way that nothing is leaked about the signature. This is done as follows: given a signature (A, e, v), the tuple $(A' : AS^{-r} \bmod n, e, v' := v + er)$ is also a valid signature on the same messages. This leads to the following protocol, as described in [Cam06],

1. $\mathcal{P}$ chooses $r \in_R \{0,1\}^{l_n + l_\phi}$, computes $A' := AS^r$, and sends A' to $\mathcal{V}$.
2. $\mathcal{P}$ executes the following protocol

$$PK\{(\varepsilon, \tilde{\nu}, \rho', \mu_i) : ZA'^{-2^{l_e - 1}} \equiv \pm A'^{\varepsilon} S^{\tilde{\nu}} \prod_i R_i^{\mu_i} \pmod{n} \wedge$$

$$\mu_i \in \{0,1\}^{l_m + l_\phi + l_c + 2} \wedge \varepsilon \in \pm\{0,1\}^{l_{e'} + l_\phi + l_c + 1}\}$$

with $\mathcal{V}$ as verifier.

where $l_\emptyset$ is sufficiently large to ensure the zero-knowledge property (e.g., $l_\emptyset = 80$), l_c is the sizes of the challenges sent by $\mathcal{V}$, $m_i \in \{0,1\}^{l'_m}$ where l'_m should be at most $l_m - l_\emptyset - l_c - 2$ and the other values are defined as in Section 4.2.

5 Service Interface

In this section, the service interface of the ideal functionality $\mathcal{F}_{\mathsf{SHC}}$ and the real-world protocol (referred to as Π_{SHC}) are described; each query is presented along with an intuitive explanation of what it means. For the formal description of $\mathcal{F}_{\mathsf{SHC}}$, see Section 6.1, and for the one of Π_{SHC}, see Section 9.3. Their properties and the adversarial model adopted in this work are also analyzed.

5.1 Interface

Secret handshake with credentials is a two-party protocol. We refer to these parties as P_i and P_j. The ideal-world adversary is referred to as $\mathcal{S}$. All parties are ITIs and own a tuple (*pid*, *sid*), where the variables *pid* and *sid* contain the PID and SID, respectively, as described in Section 4.1. (x_i, w_i) and (x_j, w_j) refer to the statements and witnesses of P_i and P_j, respectively. Note that w_i and w_j represent either a witness or the special character $\perp$, meaning that no witness is provided to the ideal functionality or the real-world protocol. In the ideal world, because the parties are dummy, all incoming queries are sent to $\mathcal{F}_{\mathsf{SHC}}$ and all outgoing queries are sent from $\mathcal{F}_{\mathsf{SHC}}$. The presented view is the view of the environment $\mathcal{Z}$. See the next subsection for queries specific to $\mathcal{F}_{\mathsf{SHC}}$.

The interface of $\mathcal{F}_{\mathsf{SHC}}$ and Π_{SHC} is as follows.

5.1.1 Incoming Queries

- Query (NewSession, P_i, P_j, x_i, w_i, *sid*) from party P_i:
  Party P_i sends this query to notify P_j that it wants to start a new secret handshake session with it, using the statement x_i and witness w_i. The Session Identifier is set to *sid*.

5.1.2 Outgoing Queries

- Private delayed query (Rejected, *sid*) to party P_i:
  This query is privately sent to P_i and notifies it that its secret handshake session with Session Identifier *sid* has been rejected.

- Private delayed query (Accepted, P_i, x_j, K_i, *sid*) to party P_i:
  This query is privately sent to P_i and notifies it that its secret handshake session with Session Identifier *sid* has been accepted and that its shared key is K_i, and confirms that the other party's witness is x_j.

5.2 Queries Specific to the Ideal Functionality $\mathcal{F}_{\mathsf{SHC}}$

5.2.1 Incoming Queries

- Query (ValidateSession, P_i, *force_reject*, K, *sid*) from $\mathcal{S}$:
  The ideal-world adversary $\mathcal{S}$ sends this query to notify $\mathcal{F}_{\mathsf{SHC}}$ that it is allowed to validate P_i's secret handshake session with Session Identifier *sid*. It means that $\mathcal{F}_{\mathsf{SHC}}$ will output either an (Accepted, ...) or a (Rejected, ...) query to P_i as described above. If P_i or P_j is corrupt, $\mathcal{F}_{\mathsf{SHC}}$ chooses K to be P_i's shared key, which is input by $\mathcal{S}$, otherwise it chooses a random key. The additional *force_reject* parameter is only taken into account if P_j is dishonest. In this case, if it is set to 1, the output is forced to (Rejected, ...). Otherwise, the behavior of $\mathcal{F}_{\mathsf{SHC}}$ is not influenced by this parameter. This is a way to model that fact that if P_j is corrupt and hence controlled by the adversary, it should be able to make P_i output (Rejected, ...) by acting in a non conforming way in the real world.

5.2.2 Outgoing Queries

- Public delayed query (SessionStart, P_i, x_i, *sid*) to party P_j:
  This query is sent to P_j as a consequence of a (NewSession, ...) query from P_i. It will notify P_j that P_i has started a new secret handshake session with it with Session Identifier *sid*.

5.3 Properties

The desired properties of our protocol and ideal functionality are those of the original secret handshake protocols [BDS+03, CJT04], which we modify to fit our new problem definition. We do it in a similar way as it is done in [ABK07], where they use the fuzzy-matching: as they do not have groups either, they also need to reformulate security properties. They are informally stated as follows.

Completeness. If two honest parties A and B with valid statement and witness tuples (x_A, w_A) and (x_B, w_B) respectively perform a handshake, then they both output (Accepted, ...), i.e. (Accepted, A, x_B, K_A, *sid*) and (Accepted, B, x_A, K_B, *sid*), respectively, provided that $\mathcal{S}$ sends the corresponding (ValidateSession, ...) queries, or that the real-world adversary doesn't drop messages.

Impersonator resistance. Adversary Adv should not be able to impersonate a party B, that owns a valid witness w_B for the statement x_B, by engaging itself in a successful handshake with an honest party A, owning a valid witness w_A for the statement x_A, while it does not hold a valid witness for the statement x_B.
More formally, adversary Adv has the right to interact with some parties of its choice and obtain their secrets, before selecting a target user B, who owns a valid witness w_B for the statement x_B. Next, Adv selects an honest user A and performs a policy negotiation step with her. They agree on statements x_B resp. x_A, for neither of which Adv owns a valid witness. Finally, Adv attempts to convince the honest party A that it holds a valid witness for statement x_B by trying to construct the correct queries to give as input to the ideal functionality. If Adv succeeds, we say that the impersonator resistance property is violated.

Detector resistance. Adversary Adv should not be able to learn whether an honest party A owns a witness w_A for her statement x_A by engaging itself in a handshake with A, while it does not have a witness for the statements chosen during the policy negotiation step.
More formally, Adv has the right to interact with some parties of its choice and obtain their secrets before selecting an honest user A. Next, they perform a policy negotiation step and agree on statements $x_{\mathcal{S}}$ resp. x_A, for neither of which Adv owns a valid witness. A random bit $b \leftarrow \{0,1\}$ is flipped: if $b = 1$, Adv interacts with A, otherwise it interacts with a random simulation. Here, a random simulation means that the ideal functionality generates random responses instead of the meaningful messages it would have output if it were interacting with A. Finally, Adv outputs a guess b* for b. We say that the detector resistance property is violated if b*=b.

Indistinguishability to eavesdroppers. An adversary who is observing (only) an ongoing handshake between two honest parties A and B cannot guess the outcome of the protocol.

As we prove in Section 10 that our secret handshake with credentials protocol securely realizes our ideal functionality $\mathcal{F}_{\mathsf{SHC}}$ for the same goal, it is sufficient to demonstrate that these properties hold for the latter. This is shown in Section 6.1.

5.4 Adversarial Model

ITMs, including adversaries, are modeled as probabilistic polynomial time (PPT) ITMs. An ITM is PPT if its total running time, in all its activations, is polynomial in the length of its input (i.e., in the number of bits written on its input tape). However, in the UC framework, an ITI can write to arbitrary tapes of other instances and new instances can be generated throughout the execution, so this definition need to be extended in order to guarantee an overall bound on a system of ITMs. This is done by requiring that the number of bits that a PPT ITM M writes onto the input tapes of other ITMs, plus the number of different ITM instances that M writes to, is less than the number of bits written on M's input tape. For more details, see Section 3.3 of the full version of [Can01].
Moreover, we adopt the static (non-adaptive) corruption model as described in the full version of [Can01]. It is defined as follows. The set of corrupt parties is fixed before the computation start, i.e., before executing the ideal-world or real-world protocol. This set remains the same until the end of the execution, when the parties have submitted their output to $\mathcal{Z}$.
This can be modeled in the UC framework by discarding any corruption message received in any activation of a party other than the first activation. As one can see, $\mathcal{F}_{\mathsf{SHC}}$ does not have a *corrupt* query. This is meant for succinctness of the ideal functionality. If the ideal-world adversary $\mathcal{S}$ wants to corrupt a party, it receives its complete internal state including its witness, if any.

6 Ideal Functionality of a Secret Handshake with Credentials

6.1 Description of $\mathcal{F}_{\mathsf{SHC}}$

The ideal functionality $\mathcal{F}_{\mathsf{SHC}}$ is a modified version of the one for key exchange in [CHK+05]. Its aim is to model the real-world secret handshake with credentials protocol Π_{SHC} in an ideal way.

Let $\mathcal{K}$ be the set of possible keys. $\mathcal{F}_{\mathsf{SHC}}$ interacts with an adversary $\mathcal{S}$ and a set of (dummy) parties (P_i and P_j) via the following queries:

Upon receipt of a query (NewSession, P_i, P_j, x_i, w_i, sid) **from party** P_i**:**
Send public delayed (SessionStart, P_i, x_i, sid) to P_j. In addition, if this is the first (NewSession, ...) query for this sid, or if this is the second (NewSession, ...) query for this sid and there is a matching record (P_j, P_i, x_j, w_j, sid), then record (P_i, P_j, x_i, w_i, sid).

Upon receipt of a query (ValidateSession, P_i, *force_reject*, K, sid) **from** $\mathcal{S}$**:**
If there is a record (P_i, P_j, x_i, w_i, sid), and this is the first (ValidateSession, ...) query for P_i with this sid, then execute one of the following steps:

- If P_j is corrupt and *force_reject*=1, then send private delayed (Rejected, sid) to P_i.
- If there is also a record (P_j, P_i, x_j, w_j, sid) such that both parties provided a valid witness, execute one of the following steps:
  - If P_i or P_j is corrupt, set $K_i \leftarrow K$.
  - Otherwise, if a key K_j has already been sent to P_j, set $K_i \leftarrow K_j$, otherwise choose $K_i \in_R \mathcal{K}$.

  Then record K_i and send private delayed (Accepted, P_i, x_j, K_i, sid) to P_i.
- Otherwise send private delayed (Rejected, sid) to P_i.

In any other case, ignore the query.

Figure 4: The ideal secret handshake with credentials functionality $\mathcal{F}_{\mathsf{SHC}}$.

6.2 Motivations for $\mathcal{F}_{\mathsf{SHC}}$

In this section, we highlight some aspects of the formulation of our ideal functionality $\mathcal{F}_{\mathsf{SHC}}$ in order to motivate the definitional choices that we made.

Unfairness. As described in the Introduction, secret handshake protocols don't provide fairness. For example if party B receives a valid response from A that lets him figure out if A had a valid credential, but never sends anything to A, the latter will not get to know if B had a witness for his statement or not. The design of the ideal functionality $\mathcal{F}_{\mathsf{SHC}}$ reflects that imperfection by allowing the adversary to choose if and when each party gets the confirmation that they hit a match. This is done by letting the adversary send the (ValidateSession, ...) queries.

Imperfections. When designing an ideal functionality, it is important that the ideal-world adversary has the same power as the real-world adversary, because only the attacks possible in the ideal world are possible in the real world. This is why the imperfections of the real-world protocol Π_{SHC} also have to be present in the ideal functionality $\mathcal{F}_{\mathsf{SHC}}$.
In the real world, when two parties are running Π_{SHC} together, the real-world adversary sees messages between these parties. Hence it can deduce that these parties are executing the protocol. Thus, we introduce the query (SessionStart, ...) sent from $\mathcal{F}_{\mathsf{SHC}}$ to parties on which the ideal-world adversary can also spy on.
Another imperfection has to be modeled. A party B that has provided a valid credential can still cheat during the second part of Π_{SHC} using the fact that the protocol is unfair and not send

a valid response to A, preventing the latter to know if B had a valid credential. Hence, A will reject while B can still accept, assuming that he had already received a valid response from A. Hence we introduce the binary parameter *force_reject* to the (ValidateSession, ...) query in $\mathcal{F}_{\mathsf{SHC}}$ to model this behavior; the adversary can make the honest party A reject at any moment if the other party B is dishonest. It implies that in both $\mathcal{F}_{\mathsf{SHC}}$ and Π_{SHC}, A cannot tell the difference between a cheating B or a honest B without a valid credential.

No key is output when the outcome is (Rejected, ...)**.** Even if parties always compute a key during the execution of Π_{SHC}, we do not claim any security on that key if the outcome of the protocol is (Rejected, ...). Hence, in the ideal world, we do not even output such a key.

Completeness. If both parties are honest and own a valid witness, they both output (Accepted, ...), i.e. (Accepted, P_i, x_j, K_i, sid) and (Accepted, P_j, x_i, K_j, sid), respectively, provided that $\mathcal{S}$ sends the corresponding (ValidateSession, ...) queries, or that the real-world adversary doesn't drop messages.

Impersonator Resistance. A party providing no valid witness will not be able to make a honest party accept by using the interface of the ideal functionality, since the latter will send a (Reject, ...) query to both parties in this case. Thus, the only way for a party to make the ideal functionality output accept is to provide a valid witness. However, in order to do this, the party will have to forge this witness. This is not possible because it implies that it would be able to guess a secret that it does not have. In the case where credentials are implemented using CL signatures, typically, one cannot forge such signatures.

Detector Resistance. The argument is almost the same as the one for the impersonator resistance. A party providing the ideal functionality with no valid witness will always receive the output (Reject, ...) and therefore cannot make a guess about the input of the other party. In order to provide a valid witness, one should forge it which is impossible, as explained above.

Indistinguishability to eavesdroppers. In the case where two honest parties are talking to the ideal functionality $\mathcal{F}_{\mathsf{SHC}}$, the ideal-world adversary (who is observing only) cannot guess the outcome of the ideal secret handshake since the ideal functionality does not leak any information. Moreover, the queries (Accepted, ...) and (Rejected, ...) are sent privately to the concerned honest parties. Claim 1 of Section 10 shows that the adversary does not get information from looking at exchanged messages during an execution of the real-world protocol, unless it breaks the DDH assumption.

7 Shared Setup Functionality $\bar{\mathcal{G}}_{\mathsf{SETUP}}$ using the GUC Framework

In this section we introduce the shared functionality $\bar{\mathcal{G}}_{\mathsf{SETUP}}$. As specified in Section 4.1, it can be accessed by any party as a part of any process. Only during the first activation does $\bar{\mathcal{G}}_{\mathsf{SETUP}}$ set the global setup data and record it internally. In all activations, it returns this data to the activating party. The latter can use this information in the ideal or the real process. $\bar{\mathcal{G}}_{\mathsf{SETUP}}$ is described in Figure 5.
We claim that the parameters $(\mathbb{G}, g, T)$ set by $\bar{\mathcal{G}}_{\mathsf{SETUP}}$ are accessible by any party (including ideal functionalities) at any time. Hence accesses to $\bar{\mathcal{G}}_{\mathsf{SETUP}}$ are implicit throughout the remainder of this work.

Upon receipt of a query (Setup, $\#shsid$) **from any party:**
If this is the first activation, set and record the following parameters:
- $\mathbb{G}$, a group in which the DDH problem is hard, with order q, where q is a large prime
- $g \in \mathbb{G}$, $g \neq 1$
- $T \in_R \mathbb{G}$, where $\log_g(T)$ is an unknown quantity

In any case output (Set, $(\mathbb{G}, g, T)$, $\#shsid$) to the activating party.

Figure 5: The shared setup functionality $\bar{\mathcal{G}}_{\mathsf{SETUP}}$.

8 Zero-Knowledge Functionality for a Relation

$\mathcal{F}_{\mathsf{ZK}}^{S}$ is a static version of the standard ideal functionality for zero-knowledge proofs for a binary relation S, as defined in the full version of [Can01]. It runs with a prover P, a verifier V and an adversary $\mathcal{S}$, and proceeds as described in Figure 6.

Upon receipt of a query (Prove, P, V, x, w, sid) **from prover** P**:**
If $(x, w) \in S$, send public delayed (Verified, P, V, x, sid) to V. Otherwise do nothing.
From now on, ignore future (Prove, ...) queries.

Figure 6: The ideal zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}^{S}$.

In our case, we need to parametrize it with the relation $\widehat{R}$ instead of S, which consists of the following:

$$\widehat{R} := \{((x, Y), (w, y)) \in L \times \mathbb{G} \times \widehat{W} \times \mathbb{Z}_q |\ ((x, w) \in R \wedge g^y = Y) \vee (w = \perp \wedge g^y = T/Y)\}$$

where
- $\widehat{W} = W \cup \{\perp\}$, the special character $\perp$ meaning that no witness is provided
- $(x, w) \in L \times \widehat{W} \subseteq R$
- L and W are the set of statements resp. witnesses
- $(x, w) \in R$ if and only if w is a valid witness for statement x

This relation is motivated in Section 9, as a part of our protocol. Moreover, to be consistent with the behavior of $\mathcal{F}_{\mathsf{SHC}}$, we introduce a new message (False, ...) which is sent to V in the situation where $(x, w) \notin R$. The difference is that we explicitly reject instead of abort, which is normally the case for zero-knowledge protocols [Can01]. We will call the resulting ideal functionality $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$. Like other ITMs, it has access to $\bar{\mathcal{G}}_{\mathsf{SETUP}}$ and is defined in Figure 7.

A note on the realization of $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$. In order to realize $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$ in the GUC framework, one would need to extend the $\bar{\mathcal{G}}_{\mathsf{SETUP}}$ shared functionality; one could use a modified version of the augmented CRS $\bar{\mathcal{G}}_{\mathsf{acrs}}$ shared setup functionality defined in [CDPW07], which would include the groups elements needed for our protocol, as defined in Section 7.
Moreover if one wants to use the adaptive corruption model, one has to be careful if prover $\mathcal{P}$ is corrupted after having committed to some value g^y, but before the challenge is sent by

Upon receipt of a query (Prove, P, V, (x, Y), (w, y), sid) **from prover** P**:**
If $((x, Y), (w, y)) \in \widehat{R}$, send public delayed (Verified, P, V, (x, Y), sid) to V. Otherwise send public delayed (False, P, V, (x, Y), sid) to V.
From now on, ignore future (Prove, ...) queries.

Figure 7: The ideal zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$.

verifier $\mathcal{V}$. In this case, simulator $\mathcal{S}$ has to produce the internal state of $\mathcal{P}$, which include g^y. However, at this point in time, $\mathcal{S}$ doesn't know if $\mathcal{P}$ has a witness or not, and thus, doesn't know if $y = \log_g(Y)$ or $y = \log_g(T/Y)$. To overcome this problem, one can use simulation-sound trapdoor commitments [MY04], which allows $\mathcal{S}$ to open the commitment in both ways using some trapdoor information.

9 A protocol for Secret Handshakes with Credentials with Access to $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$

9.1 The goal of Π_{SHC}

The goal of the our secret handshake with credentials protocol (referred to as Π_{SHC}) is to achieve a match and derive a shared key if and only if parties own specific credentials. Usual secret handshake protocols limit the choice of these credentials: parties achieve a match if and only if they own the same credential, in this case a proof of affiliation to a group, see for example [CJT04]. Our new protocol does not have this restriction as we can use arbitrary policies. On the other hand, disclosing these policies is something we cannot avoid because parties have to perform a policy negotiation step before actually running the protocol. Former secret handshake protocols can avoid this because each party starts the protocol using its group membership card as the credential. Secret handshake protocols with multiple groups exist [YT07]; however, matches are achieved only if parties are part of the exact same groups, which requires running the protocol with a sorted list of all existing groups and do one-to-one matches. *Efficient* authentication in an arbitrary condition is an open problem [KTK+08], which our protocol does not solve yet. In effect, forcing the credentials of the initiator to be related in some ways to those of the recipient would oblige them to disclose their group identities if we try to use our protocol to achieve a secret handshake in the original fashion. Hence, policies in Π_{SHC} have to be unrelated to each other.

9.2 The idea behind Π_{SHC}

The key idea of the protocol is as follows. Party A holds a credential U and $value_{1_A}$, or $value_{2_A}$. Party B holds a credential V and $value_{1_B}$, or $value_{2_B}$. A key is derived from a function of $value_{1_A}$ and $value_{1_B}$. Algebraic properties ensure that A and B cannot know both values, i.e., A cannot know $value_{1_A}$ *and* $value_{2_A}$, and B cannot know $value_{1_B}$ *and* $value_{2_B}$. This makes sure that the shared common session key is possessed by a party if and only if this party owns

a valid credential.
In the remainder of this paragraph, we describe this idea more formally. Through $\bar{\mathcal{G}}_{\mathsf{SETUP}}$, parties have access to g, an element of some group $\mathbb{G}$, and to T, a random element of $\mathbb{G}$, where $t = \log_g(T)$ is kept secret. Party A can compute a random split of T, such that $T = Y \cdot Y^*$. A can choose if she wants to know either $y = \log_g(Y)$ or $y^* = \log_g(Y^*)$, but cannot know both, as otherwise A would know t and thus break the discrete log assumption. Using the Camenisch-Stadler notation as introduced in Section 4.3, we can express the proof of knowledge that A will then perform, where R is true if w_A is a valid witness for the statement x_A:

$$PK\{(\gamma, \omega_A, \gamma^*) : ((x_A, \omega_A) \in R \wedge Y = g^{\gamma}) \vee (Y^* = g^{\gamma^*})\}$$

The protocol is symmetric, so party B will do the same with the values Z, Z^*, z, z^*, x_B and w_B as statement and witness. In the end, A and B derive their key by computing Z^y and Y^z, respectively, both equal to g^{yz}, which of course is only possible if they both own a valid witness for their respective statement.

9.3 Description of Π_{SHC}

The protocol Π_{SHC} involves two parties A and B, which are activated with inputs (NewSession, A, B, x_A, w_A, *sid*) and (NewSession, B, A, x_B, w_B, *sid*), respectively. Moreover, let $\mathcal{K}$ be the set of possible keys. Π_{SHC} has access to $\bar{\mathcal{G}}_{\mathsf{SETUP}}$ and $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$. It proceeds in two phases. First parties establish a key and, second, they proceed to a key confirmation phase. The protocol is as shown on Figure 8.
If the MAC is not received by a party after some timeout, we assume that this party will reject. In the ideal-world, this is modeled by the fact that the adversary $\mathcal{S}$ sends the (ValidateSession, ...) query with the parameter *force_reject*=1 to the ideal functionality.

10 Proving the Security of Π_{SHC}

Before showing how to construct a simulator, we would like to illustrate the situation with the two diagrams of Figure 9. On the left hand-side, the real-world adversary is implicitly represented, as it controlls B^*.

Theorem 1. *The secret handshake with credentials protocol* Π_{SHC} GUC-EUC-*securely realizes the ideal functionality* $\mathcal{F}_{\mathsf{SHC}}$ *in the* $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$*-hybrid model with static corruptions, as defined in Section 4.1.*

In order to prove this, we have to build a simulator $\mathcal{S}$ which will play the role of the ideal-world adversary, and prove that for all inputs provided by $\mathcal{Z}$ the simulation is successful. Let $\mathcal{A}$ be the real-world adversary that interacts with Π_{SHC} in the $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$*-hybrid* model. We construct $\mathcal{S}$ such that the view of any environment $\mathcal{Z}$ of an interaction with $\mathcal{A}$ and Π_{SHC} is computationally indistinguishable from its view of an interaction with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{SHC}}$.
$\mathcal{S}$ has access to a copy of $\mathcal{A}$ and simulates $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$. All messages from $\mathcal{Z}$ addressed to $\mathcal{S}$ are directly forwarded to $\mathcal{A}$ and all outputs of $\mathcal{A}$ are given back to $\mathcal{Z}$.
As Π_{SHC} is symmetric, it is sufficient to simulate one of the two parties involved in the protocol.

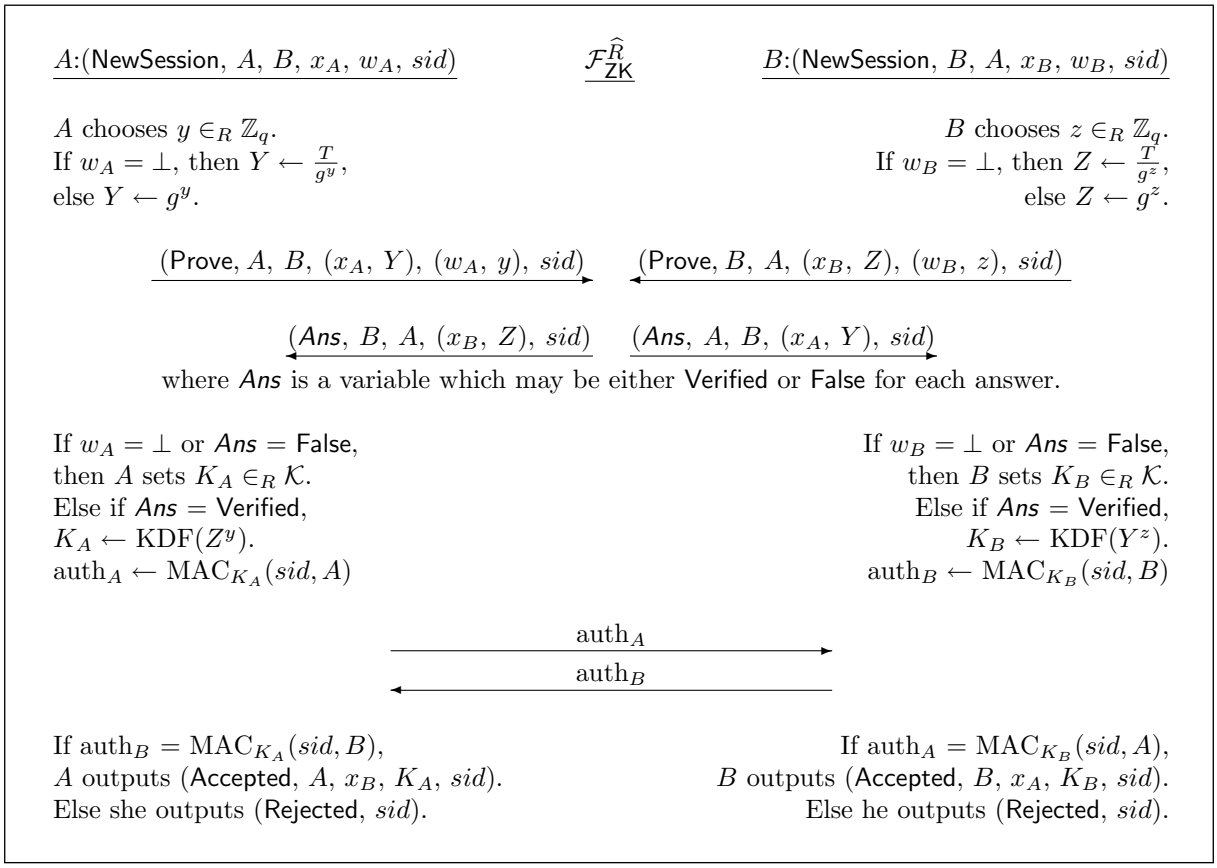
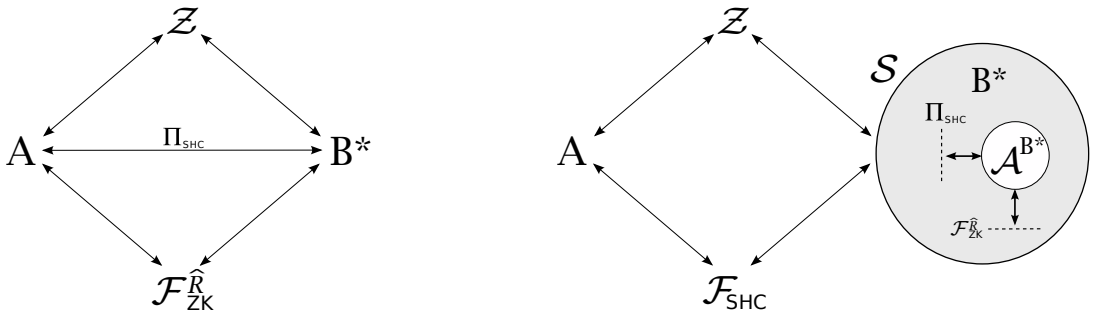


Figure 8: The secret handshake with credentials protocol Π_{SHC} realizing $\mathcal{F}_{\mathsf{SHC}}$ in the $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$-*hybrid* model.

Hence, throughout this proof, $\mathcal{S}$ plays the role of a corrupt party, which we call B^*, and interacts with honest player A. During the simulation, $\mathcal{S}$ will exchange messages with its internal copy of $\mathcal{A}$ in the name of A. We rename $\mathcal{A}$ to $\mathcal{A}^{B^*}$ as it will play the role of B^* during the simulation. The simulator has to react to four different situations depending on whether A and B^* have a witness for their respective statement or not. The core problem of the proof is that $\mathcal{S}$ does not know the state of A; if $\mathcal{S}$ discovers that B^* has no witness, $\mathcal{A}^{B^*}$'s output for Π_{SHC} will be *reject* and there is no way for $\mathcal{S}$ to figure out whether A had a witness. Hence, we have to prove that even in this case, $\mathcal{S}$ is able to achieve a simulation that is computationally indistinguishable from an execution of the real-world protocol Π_{SHC}.

Proof (Description of the simulator). At the beginning of the game, $\mathcal{Z}$ activates A and $\mathcal{A}^{B^*}$ with (NewSession, A, B^*, x_A, w_A, sid) and (NewSession, B^*, A, ${x_B}^*$, ${w_B}^*$, sid) respectively. A forwards her message to $\mathcal{F}_{\mathsf{SHC}}$, which sends (SessionStart, A, x_A, sid) to B^*. The goal of simulator $\mathcal{S}$ is to get B^*'s witness from $\mathcal{A}^{B^*}$ in order to be able to send the correct (NewSession, ...) message to also $\mathcal{F}_{\mathsf{SHC}}$.

The simulation. As the order of the two calls to $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$ is not fixed, we assume that $\mathcal{S}$ in the role of A may talk first; it is more difficult because $\mathcal{S}$ has no information from $\mathcal{A}^{B^*}$ at this point.



(a) Real World (b) Ideal World

Figure 9: Real and ideal world

$\mathcal{S}$ chooses $y \in_R \mathbb{Z}_q$ and sets $Y \leftarrow g^y$. Then $\mathcal{S}$ sends the message (Verified, A, B^*, (x_A, Y), sid) from $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$ to $\mathcal{A}^{B^*}$. Eventually $\mathcal{A}^{B^*}$ tries to query $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$ with (Prove, B^*, A, (x_{B^*}, Z), (w_{B^*}, z), sid), but the message is received by $\mathcal{S}$, which simulates $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$.

At this point $\mathcal{S}$ finds out whether $\mathcal{A}^{B^*}$ has a witness; w_{B^*} will contain the special character $\perp$ if $\mathcal{A}^{B^*}$ has no witness. In addition, $\mathcal{S}$ needs to check that $((x_{B^*}, Z), (w_{B^*}, z)) \in \widehat{R}$. If it is not, $\mathcal{S}$ sets w_{B^*} to $\perp$ in order to make A reject, just as A would do in the real world. Although B^* could still accept, this is why in the ideal world, $\mathcal{S}$ lets $\mathcal{A}^{B^*}$ decides whether it accepts.

In all cases, $\mathcal{S}$ sends (NewSession, B^*, A, ${x_B}^*$, ${w_B}^*$, sid) to $\mathcal{F}_{\mathsf{SHC}}$, which sends (SessionStart, B^*, x_{B^*}, sid) to A. Then $\mathcal{S}$ sends (ValidateSession, B^*, $force\_reject$=0, K=Y^z, sid) to $\mathcal{F}_{\mathsf{SHC}}$. Now let us differentiate these two cases:

Case 1: $w_{B^*} \neq \perp$**.** In this case, $\mathcal{S}$ can actually find out whether A has a witness from $\mathcal{F}_{\mathsf{SHC}}$'s answer. Either it has and as both parties provided a valid witness, $\mathcal{F}_{\mathsf{SHC}}$ sends (Accepted, B^*, x_A, K_{B^*}, sid) to B^*, where $K_{B^*} = Y^z$ because B^* is corrupt. Now let's finish the simulation. $\mathcal{S}$ sets $K_A \leftarrow \mathrm{KDF}(Z^y)$. In addition it computes $\mathrm{auth}_A \leftarrow \mathrm{MAC}_{K_A}(sid, A)$ and sends it to $\mathcal{A}^{B^*}$.
The latter now has three choices. Either it can send a valid auth_{B^*}, in which case the following happens. $\mathcal{S}$ sends (ValidateSession, A, $force\_reject$=0, K=Z^y, sid) to $\mathcal{F}_{\mathsf{SHC}}$, which sends (Accepted, A, x_B, K_A, sid) to A, where $K_A = Z^y$ because B^* is corrupt. Instead, if $\mathcal{A}^{B^*}$ sends an invalid auth_{B^*} or nothing at all, $\mathcal{S}$ sends (ValidateSession, A, $force\_reject$=1, K=Z^y, sid) to $\mathcal{F}_{\mathsf{SHC}}$, which sends (Rejected, sid) to A. Then in all three cases, $\mathcal{S}$ allows A forwards its output to $\mathcal{Z}$. As auth_A is valid, $\mathcal{A}^{B^*}$ sends (Accepted, B^*, x_A, K_{B^*}, sid) to $\mathcal{Z}$.

If A has no witness, $\mathcal{F}_{\mathsf{SHC}}$ sends (Rejected, sid) to B^*, which is controlled by $\mathcal{S}$. At this point, $\mathcal{S}$ knows that A does not have a valid witness because it provided $\mathcal{F}_{\mathsf{SHC}}$ with $\mathcal{A}^{B^*}$'s witness, hence $\mathcal{S}$ sets $K_A \in_R \mathcal{K}$. In addition it computes $\mathrm{auth}_A \leftarrow \mathrm{MAC}_{K_A}(sid, A)$ and sends it to $\mathcal{A}^{B^*}$. Again, $\mathcal{A}^{B^*}$ can decide to send a valid, invalid or no auth_{B^*} at all, but this is not important because, in this case, $\mathcal{S}$ is not allowed to change the output of A. In effect, according to $\mathcal{F}_{\mathsf{SHC}}$, $\mathcal{S}$ can only make A reject if B^* is corrupt, but, here, A already rejects because she has no witness. Hence, $\mathcal{S}$ sends (ValidateSession, A, $force\_reject$=0, K=K_A, sid) to $\mathcal{F}_{\mathsf{SHC}}$, which sends (Rejected, sid) to A. Then $\mathcal{S}$ allows A forwards its output to $\mathcal{Z}$. As auth_A is invalid, $\mathcal{A}^{B^*}$ rejects by sending (Rejected, sid) to $\mathcal{Z}$.

Case 2: $w_{B^*} = \perp$**.** As $\mathcal{A}^{B^*}$ did not provide a valid witness, $\mathcal{F}_{\mathsf{SHC}}$ sends (Rejected, *sid*) to B^*, which is controlled by $\mathcal{S}$. At this point, $\mathcal{S}$ doesn't know whether A has a valid witness, and, because B^* has no witness, it cannot derive K_A by computing it using a value received from $\mathcal{A}^{B^*}$, hence $\mathcal{S}$ sets $K_A \in_R \mathcal{K}$. In addition it computes $\text{auth}_A \leftarrow \text{MAC}_{K_A}(sid, A)$ and sends it to $\mathcal{A}^{B^*}$. Again, $\mathcal{A}^{B^*}$ can decide to send a valid, invalid or no auth_{B^*} at all, but this is not important because, in this case, $\mathcal{S}$ is not allowed to change the output of A for the same reason as above, except that this time, it is B^* who has no witness. Hence, $\mathcal{S}$ sends (ValidateSession, A, *force_reject*=0, K=K_A, *sid*) to $\mathcal{F}_{\mathsf{SHC}}$, which sends (Rejected, *sid*) to A. Then $\mathcal{S}$ allows A forwards its output to $\mathcal{Z}$. As auth_A is invalid, $\mathcal{A}^{B^*}$ rejects by sending (Rejected, *sid*) to $\mathcal{Z}$.

In the situation where A actually has a witness but B^* does not, $\mathcal{S}$ does not strictly follow the protocol Π_{SHC} when talking to $\mathcal{A}^{B^*}$. In effect, $\mathcal{S}$ sets K_A to a random value instead of the well-formed value it should have. Hence, we in addition want to prove that $\mathcal{A}^{B^*}$ cannot detect such a change. □

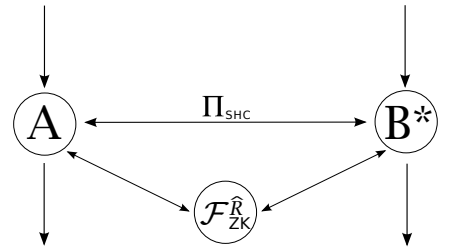
Claim 1. *$\mathcal{A}^{B^*}$, who has no valid witness, cannot distinguish between* $\text{MAC}_{K_A}(sid, A)$ *and* $\text{MAC}_{K^*}(sid, A)$ *where* K^*=$\text{KDF}(g^{yz})$*; otherwise we can use $\mathcal{A}^{B^*}$ to construct an algorithm to break the* DDH *problem defined in Section* 3.1.1.

Proof (Building an adversary breaking the DDH problem). The goal is to build an adversary $\mathcal{A}^{\text{DDH}}$ which can break the DDH assumption given an instance T, D, R of the DDH problem and access to the real-world adversary $\mathcal{A}^{B^*}$, which is able to distinguish between $\text{MAC}_{K_A}(sid, A)$ and $\text{MAC}_{K^*}(sid, A)$, but does not possess any valid witness.

$\mathcal{A}^{\text{DDH}}$ executes Π_{SHC} with $\mathcal{A}^{B^*}$. At some point, $\mathcal{A}^{\text{DDH}}$ has to feed $\mathcal{A}^{B^*}$ with the message (Verified, A, B^*, (x_A, Y), *sid*). In return, $\mathcal{A}^{B^*}$ outputs the message (Prove, B^*, A, (x_{B^*}, Z), ($w_{B^*} = \perp$, z), *sid*). Later $\mathcal{A}^{\text{DDH}}$ has to send auth_A to $\mathcal{A}^{B^*}$, who will either accept it and output 1 or reject it and output 0.

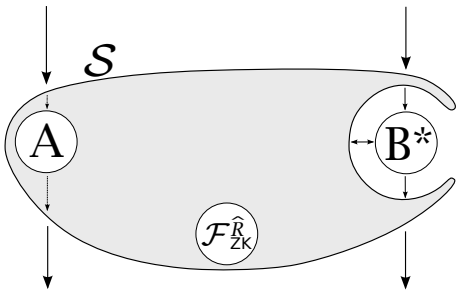
$\mathcal{A}^{\text{DDH}}$ has to choose values for Y and K, where K is the key to generate auth_A. As an intuitive assignment for Y, K we could simply use D, R and assume that T is equal to Z, which is provided by $\mathcal{A}^{B^*}$. But this does not work because Z may not be random as it is chosen by the adversary $\mathcal{A}^{B^*}$. Hence we want to make use of another random element of $\mathbb{Z}_q$, namely, t, as defined in Section 9.2. Let us do the following assignment: $Y \leftarrow D$, $Z \leftarrow \frac{T}{g^z}$, $K \leftarrow \frac{R}{Y^z}$, where $\log_g(T) = t$. Thus, $\mathcal{A}^{B^*}$ outputs 1 if and only if $K = g^{y(z-t)}$, which also means that $td = r \bmod q$. It implies that $\mathcal{A}^{B^*}$ outputs 0 in all other cases, where the key K is set to a random value because $\log_g(R)$ is random. □

Proof (Successful simulation). To prove that the simulation is successful, we have to prove that $\mathcal{Z}$ cannot distinguish whether it is interacting with the real or the ideal world, or, in our case, with the $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$-*hybrid* model or the ideal world. If the simulation fails, by Claim 1, we can build an adversary breaking the DDH problem.
To do this, we describe the series of games shown on Figure 10, in which we start with the $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$-*hybrid* model and transform it into the ideal world. On each figure, the entities A, B^* or $\mathcal{S}$ get their input from the environment $\mathcal{Z}$ and forward their output to $\mathcal{Z}$ as well. This is indicated by the vertical arrows. □



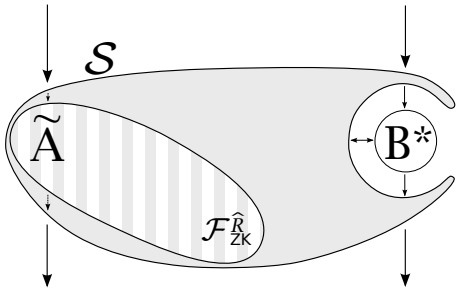
Game G_0

In the first game, we start with the $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$-*hybrid* model. A and B^* get their inputs from $\mathcal{Z}$, communicate with each other using Π_{SHC} and $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$, and send their output to $\mathcal{Z}$.



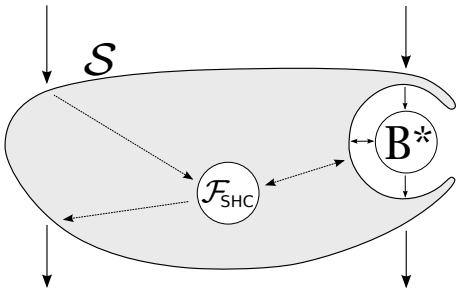
Game G_1

In the second game, all we do is introduce the simulator $\mathcal{S}$, which runs A's and $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$'s code without change. It forwards the inputs it gets from $\mathcal{Z}$ to the corresponding parties, which generate outputs that it forwards to $\mathcal{Z}$.



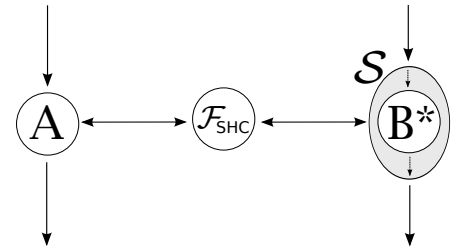
Game G_2

In the third game, we change the code of A, hence renamed $\tilde{A}$, so that she now chooses her key K_A at random, independently of whether her or B^*'s witness is valid. The latter means $\tilde{A}$ no longer follow Π_{SHC}. However, by claim 1, B^* cannot detect this change unless he can break the DDH assumption.



Game G_3

In the fourth game, we introduce $\mathcal{F}_{\mathsf{SHC}}$, which replaces $\tilde{A}$ and $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$ with the same functionality.



Game G_4

In this step, the simulator $\mathcal{S}$ does not look at A's witness any more; A as a dummy party is introduced. $\mathcal{S}$ leaves to $\mathcal{F}_{\mathsf{SHC}}$ the task of generating the outputs accordingly. Hence, finally, we have built our simulator $\mathcal{S}$, and end up with the ideal world.

Figure 10: From $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$-*hybrid* model to ideal world

11 Using CL Signatures instead of (x, w) Pairs

In order to make our scheme more practical, we can use CL signatures [CL02] instead of (x, w) pairs. In Section 4.2, the signature scheme was briefly outlined. This can be done as follows. Signatures are generated by accredited authorities. Prior to the execution of the protocol, a policy agreement step as described in the Introduction is performed between the two parties in order to negotiate policies. The outputs of this step are two sets of values labeled $(c_0^A, \ldots, c_{L-1}^A)$ and $(c_0^B, \ldots, c_{L-1}^B)$, which correspond to the policy accepted for each party. Later parties will have to prove that their credentials satisfy these policies. Possible proofs are described in Section 4.3.
In this variant, statements consist of public values of a CL signature and a policy, and witnesses consist of the corresponding CL signature on a message and the message itself. The following changes are made to the scheme:

- In protocol Π_{SHC} and in the ideal functionalities $\mathcal{F}_{\mathsf{SHC}}$ and $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$, the statements labeled x or similar are replaced by tuples labeled $x_{CL} := (S, R_0, \ldots, R_{L-1}, n, Z, c_0, \ldots, c_{L-1})$ or similar.
- In protocol Π_{SHC} and in the ideal functionalities $\mathcal{F}_{\mathsf{SHC}}$ and $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$, the witnesses labeled w or similar are replaced by tuples labeled $w_{CL} := (A, e, v, m_0, \ldots, m_{L-1})$ or similar.
- In $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$ and $\mathcal{F}_{\mathsf{SHC}}$, the check that (x, w) pairs are valid, also known as relation R, is replaced by a check that the given tuple (x_{CL}, w_{CL}) is a valid CL signature (this is described in section 4.2) and that the m_i fulfill the policy given by the c_i, using the appropriate zero-knowledge proofs of representation.

The reason for choosing the CL signature scheme is the following. As the UC design of a complex protocol is modular, it is possible to realize each ideal functionality by a separate protocol as explained in section 4.1. Further work could include the realization of $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$ by a UC-zero-knowledge protocol in which a party would prove possesion of a credential to another one. It is known how to do this in an efficient way when using CL signatures as credentials as explained in section 4.3.

12 Conclusion

We have introduced a more general protocol for secret handshakes which allows matching on credentials, and demonstrated that it is secure in the GUC framework using the hybrid model. It implies that security is maintained under composition with other secure protocols in the same model, even when running concurrently with any number of arbitrary protocols controlled by the adversary. In order to achieve this, we have proposed the first version of an ideal functionality for secret handshakes with credentials.

We would like to conclude on a number of points that can be implemented in future work.

Anonymity in the UC framework. In many cases, it is essential that parties performing a secret handshake are anonymous, otherwise their identity could leak information about the credentials they own and, thus, about the outcome of the protocol. This is particularly true for original secret handshake protocols, where what is being proven is group membership. In this case *who* parties are, i.e, their identities, have to be kept secret because it could leak information about the fact that they are part of some group. However, the UC framework defines a PID (Party Identifier) that has to be included in every message sent by parties and that contains the identity of the sending party. A possible way to add anonymity to the UC framework would be to design an ideal functionality providing (possibly unlinkable) pseudonyms to be used in the PID field instead of the real identity of parties.

Realizing the zero-knowledge ideal functionality $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$. Our new protocol makes access to a zero-knowledge ideal functionality for a specific relation $\widehat{R}$, thus running in the hybrid model. Future work could consist in designing a zero-knowledge protocol realizing $\mathcal{F}_{\mathsf{ZK}}^{\widehat{R}}$.
Furthermore this protocol should be able to allow a prover to prove possession of a CL signature to a verifier using a proof of knowledge. In order to do this, the protocol described in Section 4.3 could be used.

Modeling the issuer. One could design a new protocol in order to include the issuer of credentials into the system, making it more complete. It could be done by extending the ideal functionality or by using the existing one as a modular component, hence taking advantage of the UC framework composition theorem.

Integrating the policy negotiation step. One could design a protocol that includes the policy negotiation step, in such a way that no information about the statements would be shared with the other party if no valid witness is provided for that statement. This would allow credentials to be related to each other.

Multi-party secret handshakes secure in the UC framework. Researchers have been working on multi-party secret handshakes [TX06, JKT06, JKT07], but there is no such work proven secure in the UC framework yet.

References

[ABK07] Guiseppe Ateniese, Marina Blanton, and Jonathan Kirsch. Secret handshakes with dynamic and fuzzy matching. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium, NDSS*, pages 159–177, San Diego, California, USA, February 2007. The Internet Society.

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *In Proceedings of the 24th IEEE Symposium on Security and Privacy*, pages 180–196, Oakland, CA, May 2003.

[BM92] Steven Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 72–84, May 1992.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security - CCS '93*, pages 62–73, New York, NY, USA, 1993. ACM Press.

[Cam06] Jan Camenisch. *Cryptographic Protocols*, chapter Direct Anonymous Attestation Explained. Addison-Wesley, 2006.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001. Full version available at the Cryptology ePrint Archive, Report 2000/067, `http://eprint.iacr.org/2000/067`.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Proceedings of the 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer Berlin / Heidelberg, May 2007. Full version available at the Cryptology ePrint Archive, Report 2006/432, `http://eprint.iacr.org/2006/432`.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer Berlin / Heidelberg, May 2005.

[CJT04] Claude Castelluccia, Stanislaw Jarecki, and Gene Tsudik. Secret handshakes from CA-oblivious encryption. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 293–307. Springer Berlin / Heidelberg, December 2004.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Guiseppe Persiano, editors, *3rd Conference on Security in Communication Networks - SCN '02*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Berlin / Heidelberg, September 2002.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burton Kaliski, editor, *Advances in Cryptology - CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 410–424. Springer Berlin / Heidelberg, August 1997.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer Berlin / Heidelberg, July 1998.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690 –728, July 1991.

[JKT06] Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Authentication for paranoids: Multi-party secret handshakes. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security*, volume 3989 of *Lecture Notes in Computer Science*, pages 325–339. Springer Berlin / Heidelberg, July 2006.

[JKT07] Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Group secret handshakes or affiliation-hiding authenticated group key agreement. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 287–308. Springer Berlin / Heidelberg, February 2007.

[JKT08] Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Beyond secret handshakes: Affiliation-hiding authenticated key exchange. In Tal Malkin, editor, *Topics in Cryptology - CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 352–369. Springer Berlin / Heidelberg, April 2008.

[JL07] Stanislaw Jarecki and Xiaomin Liu. Unlinkable secret handshakes and key-private group key management schemes. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 270–287. Springer Berlin / Heidelberg, June 2007.

[KTK+08] Yutaka Kawai, Shotaro Tanno, Takahiro Kondo, Kazuki Yoneyama, Noboru Kunihiro, and Kazuo Ohta. Extension of secret handshake protocols with multiple groups in monotone condition. In Kyo-Il Chung, Kiwook Sohn, and Moti Yung, editors, *Information Security Applications*, volume 5379 of *Lecture Notes in Computer Science*, pages 160–173. Springer Berlin / Heidelberg, September 2008.

[MY04] Philip MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 382–400. Springer Berlin / Heidelberg, April 2004.

[TX06] Gene Tsudik and Shouhuai Xu. A flexible framework for secret handshakes. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies*, volume 4258 of *Lecture Notes in Computer Science*, pages 295–315. Springer Berlin / Heidelberg, December 2006.

[XY04] Shouhuai Xu and Moti Yung. k-anonymous secret handshakes with reusable credentials. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security - CCS 2004*, pages 158–167, Washingtion, DC, USA, October 2004. ACM.

[YT07] Naoyuki Yamashita and Keisuke Tanaka. Secret handshake with multiple groups. In Jae-Kwang Lee, Okyeon Yi, and Moti Yung, editors, *Information Security Applications*, volume 4298 of *Lecture Notes in Computer Science*, pages 339–348. Springer Berlin / Heidelberg, August 2007.

A Secret Handshakes Security Games

For completeness, we include the security games of the original secret handshake scheme defined in [BDS+03]. They are slightly reformulated so that the reader can understand them without prior reading of [BDS+03].

Impersonator resistance. Intuitively, the impersonator resistance property is violated if an honest party V, who is a member of group G, authenticates an adversary Adv as a group member, even though Adv is not a member of G.
More formally the notion of impersonator resistance is modeled by the following game:

1. Adversary Adv interacts with some users of its choice and obtains secrets of some users $U_i \notin G$.
2. Adv selects a target user $V \in G$.
3. Adv attempts to convince the honest party V that Adv $\in G$ by trying to construct the correct responses in the secret handshake protocol.

Adversary Adv wins the game if it performs a successful handshake with V.

Figure 11: The impersonator resistance security game.

Detector resistance. Intuitively, an adversary Adv violates the detector resistance property if it can decide whether some honest party V is a member of some group G, even though Adv is not a member if G.
More formally the notion of detector resistance is modeled by the following game:

1. Adversary Adv interacts with some users of its choice and obtains secrets of some users $U_i \notin G$.
2. Adv selects a target user $V \in G$.
3. A random bit $b \leftarrow \{0, 1\}$ is flipped.
4. If $b = 1$, Adv interacts with V, otherwise it interacts with a random simulation.
5. Adv outputs a guess b* for b.

Adversary Adv wins the game if b*=b.

Figure 12: The detector resistance security game.

Here, a random simulation means that random responses are generated instead of the meaningful messages V would send to Adv.