A Cryptographic Analysis of OPACITY

Özgür Dagdelen Marc Fischlin Tommaso Gagliardoni
Giorgia Azzurra Marson Arno Mittelbach Cristina Onete

Darmstadt University of Technology, Germany
www.cryptoplexity.de
oezguer.dagdelen@cased.de marc.fischlin@gmail.com
tommaso.gagliardoni@cased.de giorgia.marson@cased.de
arno.mittelbach@cased.de cristina.onete@gmail.com

Abstract. We take a closer look at the Open Protocol for Access Control, Identification, and Ticketing with privacY (OPACITY). This Diffie–Hellman-based protocol is supposed to provide a secure and privacy-friendly key establishment for contactless environments. It is promoted by the US Department of Defense and meanwhile available in several standards such as ISO/IEC 24727-6 and ANSI 504-1. To the best of our knowledge, so far no detailed cryptographic analysis has been publicly available. Thus, we investigate in how far the common security properties for authenticated key exchange and impersonation resistance, as well as privacy-related properties like untraceability and deniability, are met.

OPACITY is not a single protocol but, in fact, a suite consisting of two protocols, one called Zero-Key Management (ZKM) and the other one named Fully Secrecy (FS). Our results indicate that the ZKM version does not achieve even very basic security guarantees. The FS protocol, on the other hand, provides a decent level of security for key establishment. Yet, our results show that the persistent-binding steps, for re-establishing previous connections, conflict with fundamental privacy properties.

1 Introduction

OPACITY is short for the Open Protocol for Access Control, Identification, and Ticketing with privacY. It is a Diffie–Hellman-based protocol to establish secure channels in contactless environments. According to Eric Le Saint of the company ActivIdentity, co-inventor in the patent application [57], the development has been sponsored by the US Department of Defense [58]. The inventors have declared the contributions to OPACITY to be a statutory invention with the United States Patent and Trademark Office, essentially allowing royalty-free and public usage of the contribution. The protocol has been registered as an ISO/IEC 24727-6 authentication protocol [33] and is specified in the draft ANSI 504-1 national standard (GICS) [30]. Informal yet outdated descriptions are available through the homepage of the Smart Card Alliance [3].[1]

1.1 Security Assessment of OPACITY

As Eric Le Saint emphasizes in his description of OPACITY [58], "This protocol was designed expressly to remove the usage restrictions on contactless transactions while still delivering high

[1] We stress that none of the authors of the present paper has been involved in the development of OPACITY, or is employed by ActivIdentity, or is supported by a non-academic governmental agency for conducting this research.

performance security and privacy." Surprisingly, we are not aware of any profound and public cryptographic analysis of the protocol, including clear claims about security and privacy goals. The best effort, in terms of the Smart Card Alliance, seems to be compliance with standards [3]:

> "The protocol strictly follows U.S. government and international standards. It has been assessed for compliance with the NIST standard for key establishment protocols (SP 800-56A). As a consequence, further protocol design reviews are unnecessary prior to FIPS 140-2 security certification."

It is of course not the case —and we do not think that the Smart Card Alliance statement suggests so— that compliance with SP 800-56A, or certification according to FIPS 140-2, instantaneously gives strong cryptographic security guarantees. The NIST document SP 800-56A [50] only provides useful but, nonetheless, high-level *recommendations* for key-establishment schemes based on the discrete logarithm problem, and specifies some schemes from ANSI X9.42. To the best of our knowledge, it has not been shown formally yet under which conditions protocols complying with SP 800-56A are also cryptographically secure (in whatever sense). This is particularly true as OPACITY supports renegotiation techniques and also states privacy enhancement as an additional goal. Neither property is discussed in SP 800-56A.

Similarly, even if OPACITY was FIPS 140-2 certified and thus checked by an accredited authority, this does not necessarily imply strong security guarantees either. An obvious testimony to this argument are the easy attacks on FIPS 140-2 level 2 certified USB memory tokens where access was always granted for a fixed string, independently of the password [20, 21]. Certification according to FIPS 140-2, and this is acknowledged in the standard, only intends "to maintain the security provided by a cryptographic module" in the utilized environment; the "operator of a cryptographic module is responsible for ensuring that the security provided by the module is sufficient." (see [48]).

Hence, we believe that OPACITY deserves a closer cryptographic look. Clearly, there are many practical protocols which lack such an analysis, or have at least not been scrutinized publicly. What makes OPACITY a worthwhile object for a cryptographic analysis is:

- OPACITY is standardized and may thus be deployed extensively in the near future. This is all the more true as it is a general purpose protocol, suitable, for instance, for use in access control for buildings, but also for ticketing in transport systems [58].
- OPACITY does not seem to be deployed broadly yet. It is our firm belief that protocols should be rigorously analyzed *before* they are actually employed, in order to prevent damage caused by weaknesses discovered after deployment. Furthermore, patching a popular protocol in use is often intricate and progresses slowly (see the example of MD5-based certificates [61]).
- OPACITY still has a decent level of abstract description complexity. While nonetheless being quite complex underneath, especially taking into account different execution modes such as renegotiation steps (called persistent binding for OPACITY), this should be contrasted with similar protocols like SSL/TLS where conducting cryptographic proofs is tedious; such works often focus on particular parts or (modified) versions of the protocol [26, 46, 55, 35].

Another point, which we initially thought speaks for OPACITY, is the availability of an open source implementation on Source Forge [52]. Unfortunately, as later confirmed by the developers of OPACITY [59], this implementation seems to refer to an outdated version. The differences were sufficiently large such that we did not investigate the source code in detail about the

realization of the cryptographic concepts; nonetheless, we occasionally consulted the source code in order to extrapolate, in case some specification details were missing.

1.2 Our Results

OPACITY is a family of key-exchange protocols based on Elliptic Curve Cryptography. It comes in two versions, called Zero-Key Management (O-ZKM) and Full Secrecy (O-FS). The first name is due to the fact that the terminal does not need to maintain registered public keys. As such, the parties in the O-ZKM protocol run a Diffie–Hellman based key-exchange protocol using an ephemeral key on the terminal's side and a static (presumably on-card generated) key for the card. The experienced reader may immediately spot the weakness in this approach: since the terminal only uses ephemeral keys, anyone can in principle impersonate the terminal and successfully initiate a communication with the card! Before we go into further details of the security of the protocols, let us point out that the second protocol, O-FS, uses long-term keys on both sides and runs two nested Diffie–Hellman protocols, each one with the static key of the parties and an ephemeral key from the other party. This at least rules out obvious impersonation attacks.

Targeted Security Properties. Obviously, OPACITY aims at establishing a secure channel between the parties and to provide some form of entity authentication, especially impersonation resistance against malicious cards. Yet, at the same time, OPACITY also seems to target privacy properties. There seems to be a general and rough agreement what we expect from a "secure" key-exchange protocol, despite technical differences in the actual models [5, 16]. We opted for the common Bellare-Rogaway (BR) model for key exchange but we also consider key-compromise impersonation resistance and leakage of ephemeral secrets in the related eCK model [41]. We note that cryptographic analysis of similar key exchange protocols, such as for NIST's KEA [4, 42, 39] or for the ANSI X9.63 specified UM protocols [4, 44] cannot be transferred to OPACITY, as these protocols differ in security-relevant details and do not support renegotiation (and do not touch privacy issues); we comment on the differences within.

The privacy requirements for OPACITY are, however, less clear than the ones for key secrecy. This is all the more true as they are never specified in the accompanying documents. An earlier version of the OPACITY protocol description [60] mentions the following two goals for the O-FS protocol:

- "The OPACITY protocol does not divulge any data that allows the correlation of two protocol executions with same ICC [card] during an OPACITY session."
- "The OPACITY protocol does not divulge any identifier associated to a particular ICC or card holder during an OPACITY session."

The first requirement resembles the well-known notion of *untraceability* for protocols. We adopt the framework of Ouafi and Phan [53] which can be seen as a "BR-like" definition of the Juels and Weiss model [36], matching our approach for the key-agreement part. While the renegotiation steps build into the protocol allow for a simple attack on untraceability we can show that, if we remove these steps from the protocol O-FS does, in fact, achieve untraceability. For O-ZKM, on the other hand, even without renegotiation untraceability is not fulfilled.

The second desirable privacy property seems to be weaker in that it allows linkability in principle, but tries to hide the card's or the card holder's identity. We therefore introduce a notion called *identity hiding* which also follows the BR attack model, but instead of preventing the adversary from distinguishing between two cards —as for untraceability— we only guarantee that one cannot deduce the card's certificate (i.e., its identity).

	OPACITY-ZKM	OPACITY-FS
BR key secrecy	(only passive and if modified)	✓
+ forward secrecy	—	(only weak)
Impersonation Resistance	(only cards)	(only cards)
Untraceability	—	(only w/o persistent binding)
Identity Hiding	—	✓
(Outsider) Deniability	(only w/o persistent binding)	(only w/o persistent binding)

Table 1: Security properties of the OPACITY protocol

Basically, identity hiding is similar to recognizing a person without knowing the person's name. By contrast, untraceability is similar to not being able to tell that a particular person has been seen twice (this is independent of a person's name). Clearly, identity hiding gives weaker anonymity guarantees than untraceability or anonymity of credential systems [10, 17]. Even direct anonymous attestation [11] or cross-domain anonymity as in the case of the German identity card [6] support linkability only within specified domains but are otherwise untraceable. Hence, the notion of identity hiding should be taken with caution.

Another desirable privacy property for OPACITY may be *deniability* [25], that is, the inability to use transcripts of communications as proofs towards third parties. Although not explicitly listed as a goal, it may be advantageous for a multi-purpose card protocol like OPACITY. There are different approaches and levels of deniability [9, 23, 24, 22, 28]; in light of what OPACITY can achieve we focus on a very basic level protecting only against abuse of transcripts between honest parties (denoted *outsider deniability* here).

Finally, the goal of the OPACITY protocols is to establish a key which is subsequently used to secure communication between the card and the terminal. As such, one is of course interested in the security of the secure messaging protocol of OPACITY as well as in the overall composition of the key-agreement protocol and the secure messaging. Here, we rely on recent results for the secure composition of BR-secure key-exchange protocols [14, 12]. We next discuss and illustrate exactly which security levels are achieved by OPACITY.

Achieved security properties. Our results are summarized in Table 1. The protocol O-ZKM cannot achieve BR-security against malicious terminals. Even for passive adversaries (which can only observe executions between honest parties) the protocol is not known to be secure; it *does* fulfill BR-security *only* after a slight modification of the protocol. While this modification is provably necessary to conduct a security proof in the BR model, we note that the O-ZKM standard seems to contain further security-critical ambiguities and functional misspecification, which we conservatively interpreted to the advantage of the protocol (see Appendix A.1). The O-FS protocol achieves BR-security under the Gap Diffie–Hellman assumption [51] in the random-oracle model, assuming that the underlying cryptographic primitives are secure.[2] As for impersonation resistance, since the terminal does not authenticate towards the card, we can only hope to achieve security against malicious cards. This is met for both protocols given that the underlying message authentication scheme is secure.

As far as privacy is concerned, we show that neither protocol achieves untraceability nor even a weakened form of untracebility. For O-ZKM this is quite clear, as parts of the card's certificate are sent in clear. For O-FS the certificate is encrypted, yet we show that it is easy to desynchronize the cards' states and hence, due to persistent binding, to mount privacy attacks via desynchronization attacks. If, on the other hand, we only consider O-FS without renegotiation (and thus without any accumulated state), untraceability is met. Note that this

[2]This apparently innocent assumption about the security of the primitives has a hidden layer underneath. OPACITY is not fully specified in the standards and operates in some arguably doubtful modes, so this assumption must be taken with caution. We comment on this later.

is not the case for O-ZKM, that is, even without persistent binding (i.e., renegotiation) O-ZKM is traceable.

For identity hiding, we can show that it is met by O-FS but not by O-ZKM. However, let us note that O-ZKM contains steps which indicate that some form of identity hiding was aimed for: parts of the identity are only sent encrypted. Nevertheless an easy attack exists.

Concerning (outsider) deniability, we again only give a conditional result: OPACITY without persistent binding can be proved (outsider) deniable both for O-FS and O-ZKM. Persistent binding does, however, allow for simple attacks in many of the existing models for deniability, as well as, in our rather weak model of outsider deniability. Furthermore, persistent binding opens the door to physical attacks, for example, by simply comparing the state of the physical registers containing the persistent binding information of a terminal and card, one could extract high-confidence proofs that the card and terminal have been partnered in at least a single session.

2 Security Model

We define five major desirable security properties for the OPACITY protocol. The first one is the standard Bellare and Rogaway [5] notion for authenticated key-exchange, which captures the fact that the derived keys are secure. The second property is (card) impersonation resistance, mainly useful for access control applications (at metro stations, buildings, etc.). The last three properties refer to privacy: untraceability guarantees that cards cannot be tracked by outsiders; identity-hiding ensures that card certificates (or identifiers) remain private from outsiders; and deniability ensures that communication transcripts cannot be used as a proof towards third parties.

2.1 Key Secrecy (Authenticated Key Exchange)

We analyze OPACITY with respect to key secrecy in the real-or-random security model by Bellare and Rogaway [5]. Roughly speaking, an adversary should not be able to tell apart a genuine session key from a uniformly sampled key from the key space. The security model defines so-called sessions, describes an attack model, and shows a winning condition for an adversary.

Sessions. The model considers a set of honest participants, which we call terminal or card in the OPACITY scenario. At the outset, participants are associated with a long-term key pair $(\mathsf{sk}, \mathsf{pk})$ and we assume that the public keys pk are certified by a certification authority $\mathcal{CA}$. In particular, the $\mathcal{CA}$ checks the well-formedness of the key (e.g., that the keys are generated correctly). However, parties need not provide a proof of knowledge of the corresponding secret key to the $\mathcal{CA}$. Each participant may run several instances of the key agreement protocol Π concurrently; we call the j-th instance of party P by P_j. In order to derive a session key, protocol Π is executed between two instances of the corresponding parties. An instance is called *initiator* (resp. *respondent*) if it sends the first (resp. second) message in the protocol. In OPACITY, the initiator is always the terminal and the respondent is always the chip card. We do not restrict parties to be of either type, though, i.e., any party can potentially act as either terminal or card.

Upon successful termination of Π we assume that the participating instance P_i outputs a session key k, the session identifier sid, and a partner identifier pid of the intended partner. For O-FS the partner identity is determined through (the identity in the) certificate. The O-ZKM

version has no terminal authentication, and so the card cannot identify the partner; here the pid field is left empty.

Attack Model. Each user instance can be accessed by an adversary, which we denote $\mathcal{A}$, by means of an oracle providing the interface of the protocol instance. The adversary has full control of the network and schedules the delivery of possibly tampered messages. Initially, $\mathcal{A}$ receives all (registered) public keys of all participants, also called *honest* users. Afterwards, the adversary may register parties with arbitrarily chosen public keys, and since we do not assume proofs of knowledge in the registration phase, adversarial public keys may even be identical to the keys of honest users. Parties for which the adversary registers a key are called *adversarially controlled* or *malicious*, alternatively.

The adversary may query the following oracles:

$\mathsf{Init}(P, i, \mathsf{par})$ initializes a session i of honest user P for parameters par. This command makes party P_i automatically the initiator of a subsequent protocol run. In OPACITY, par indicates if the terminal should start a re-negotiation or not and this choice is, thus, under full control of the adversary. Similarly, for a card the parameter par indicates if the card can run re-negotiation in principle.

$\mathsf{Execute}(P, i, \mathsf{par}, P', j)$ causes the honest users P and P' to run the protocol Π for (fresh) instances i and j, where P_i acts as the initiator and takes parameters par. The final output is the transcript of a protocol execution. This query simulates a passive attack where the adversary merely eavesdrops the network.

$\mathsf{Send}(P, i, m)$ causes the instance i of honest user P to proceed with the protocol upon receiving message m. The output is the message generated by P for m and depends on the state of the instance. This query simulates an active attack of the adversary where the adversary pretends to be the partner instance.

$\mathsf{Reveal}(P, i)$ returns the session key of the input instance. The query is answered only if the session key was generated and the instance has terminated in accepting state and the user is not controlled by the adversary. This query models the case when the session key has been leaked. We assume without loss of generality that the adversary never queries about the same instance twice.

$\mathsf{Corrupt}(P)$ enables the adversary to obtain the party's long-term key sk and its internal state st, e.g., for persistent binding, but without the ephemeral session values. This is the so-called *weak-corruption* model. In the *strong-corruption* model the adversary also obtains in addition the state information including random coins and ephemeral values of all instances of user P. The corrupt queries model complete user compromise and allow to model forward secrecy. Since we analyze stateful protocols, we have to also account for the internal (persistent) state of party P, apart from any long-term secrets. Upon corruption, user P and all its instances are considered to be adversarial controlled.

$\mathsf{Test}(P, i)$ is initialized with a random bit b. Assume the adversary makes a test-query about (P, i) during the attack and that the instance has terminated in accepting state, holding a secret session key k. Then, the oracle returns k if $b = 0$ or a random key k' from the domain of keys if $b = 1$. If the instance has not terminated yet or has not accepted or the user is adversarially-controlled, then the oracle returns $\perp$. This query should determine the adversary's success to tell apart a genuine session key from an independent random key. We assume that the adversary only makes a single Test-query during the attack, even if we deal with stateful executions; a hybrid argument extends this to multiple test-queries.

$\mathsf{Register}(P^*, \mathsf{pk}^*)$ allows the adversary to register a public key pk^* in the name of a new user (identity) P^*. The user and all its instances are immediately considered to be adversarially controlled.

In addition, since we work in the random oracle model, the attacker may also query a random hash function oracle.

We assume that the adversary always knows if an instance has terminated and/or accepted. This seems to be inevitable since the adversary can send further messages to check for the status. We also assume that, for accepting runs, the adversary learns session and partner identifiers.

We consider here active adversaries who might run protocol instances concurrently. This is modeled by giving the adversary access to the $\mathsf{Send}()$ oracle. A passive adversary is prohibited to query $\mathsf{Send}()$ but, instead, may receive transcripts of Π instances via $\mathsf{Execute}()$. The goal of an adversary is to distinguish genuine session keys from random ones in a so-called *fresh* instance, as defined below. This is formalized by giving the adversary access to the Test-oracle.

Partner, Correctness, and Freshness. We call two instances P_i and P'_j *partnered* if both have terminated in an accepting state, and they share the same sid. In this case, the pid indicates the intended partner. Correctness of a key agreement protocol Π requires that participants in an honest, untampered execution of Π output the same session key, that each pid points to the corresponding partner, and that the participating instances are then partnered.

In order to define the security of an authenticated key exchange (AKE) we require the notion of freshness. We call an instance P_i *fresh*, if there has been (i) no $\mathsf{Reveal}(P, i)$ query at any point, (ii) no $\mathsf{Reveal}(P', j)$ where P'_j is partnered with P_i, (iii) no $\mathsf{Corrupt}(\cdot)$ queries, and (iv) neither P_i nor a partner of P_i is adversarially-controlled.

When we consider forward secrecy, we allow $\mathsf{Corrupt}(P)$-queries if there was no previous $\mathsf{Test}(P, i)$-query or if so, then the instance P_i must be involved in one $\mathsf{Execute}(\cdot)$-query only and, in particular, no $\mathsf{Send}(P, i, m)$-query. Instances which are fresh with respect to forward secrecy are called *fsec-fresh*. This notion basically means that even if a party loses its long-term secrets and/or its internal state has leaked, the session keys of previous executions (before the disclosure) of the protocol Π remain private. Even future executions of honest parties (using possibly ephemeral hidden values) maintain secure. We also introduce *weak* forward secrecy where we relax forward secrecy by disallowing the adversary to query $\mathsf{Corrupt}(P)$ before a $\mathsf{Test}(P, i)$-query. This rules out that future executions between honest parties remain private. Thus, only executions of honest parties before corruptions remain secure.

On a Single Test-query. In general it suffices to consider only a single Test-query, since the case for multiple queries for many sessions follows by a hybrid argument [1], decreasing the success probability of an adversary by a factor equal to the number of queries made. Even only stated for stateless KE protocols, their arguments easily extend to stateful KE protocols since the state is never revealed unless a $\mathsf{Corrupt}$-query is made. However, this execution cannot considered fresh anymore.

AKE Security. Eventually, the adversary $\mathcal{A}$ outputs a guess b' for the secret bit b used in the Test-oracle. The adversary is *successful* iff: $b = b'$, and the instance P_i in the Test-oracle is fresh (resp. [weakly] fsec-fresh). We are interested in the advantage of the adversary over the simple guessing probability of $1/2$. We usually consider security relative to the adversary's parameters, such as its running time t, the number q_e of initiated executions of protocol instances of Π, and, modeling the key derivation function as a random oracle, the number q_h of random oracle queries of the adversary. For some of the security notions we also make the number of Test queries explicit through a parameter q_t.

Definition 2.1 (Key Secrecy) *We call a protocol* Π*, running for security parameter* λ*,* (t, q_e, q_h, ϵ)*-secure if no algorithm running in time* t*, invoking* q_e *instances of* Π *and making at most* q_h *queries to the random oracle can win the above experiment with probability greater than* $\frac{1}{2} + \epsilon$*. We call the value* $\left|\Pr[\mathcal{A} \textsf{ wins}] - \frac{1}{2}\right|$ *the* advantage *of the algorithm* $\mathcal{A}$*, and we denote the maximum over all* (t, q_e, q_h)*-bounded* $\mathcal{A}$ *by* $\mathbf{Adv}^{ake}_{\Pi}(t, q_e, q_h)$*.*

Analogously, we denote by $\mathbf{Adv}^{\text{ake-fsec}}_{\Pi,\mathcal{A}}$ resp. $\mathbf{Adv}^{\text{ake-wfsec}}_{\Pi,\mathcal{A}}$ the advantage of algorithm $\mathcal{A}$ when considering forward secrecy resp. weak forward secrecy.

Further Desirable Security Properties. The BR model is a strong security model providing confidentiality of agreed session keys and authenticity (i.e., at most a partner will hold the same derived keys). In addition, one can show forward secrecy when adapting the freshness notion. However, as pointed out by LaMacchia et al. [41], some attack scenarios are not considered in the BR model. For this reason, LaMacchia et al. [41] proposed the eCK model, an extension of the AKE model by Canetti and Krawczyk [16], whose additional security properties follows.

Key-Compromise Impersonation (KCI). Here, an adversary cannot impersonate herself to a party even if she is in possession of this party's long-term secret.

Leakage of ephemeral secrets. Here the model allows leakage of internal information of a session, i.e. even if the adversary learns the randomness used in a given execution and can therefore compute the ephemeral keys, the corresponding session key remains confidential.

2.2 Impersonation Resistance

The notion of authenticated key exchange ensures that only a partner can compute the session key. For some application scenarios, however, we may also need that the terminal can be sure of the card's identity. This could be guaranteed by subsequent use of the computed session keys, but this is application-dependent. Impersonation resistance, as defined here, gives instead direct guarantees and is closer to the security of identification schemes. We give a strong definition based on the BR framework, which includes common properties like passive and active security for identification schemes. Still, note that we only consider impersonation by malicious cards (not terminals).

The attack model resembles AKE, but this time there are no Test-queries. The adversary's goal is to impersonate an honest card, without using trivial Man-in-the-Middle relaying attacks or making the terminal accept a card which has not been issued (resp. certified) by the certification authority $\mathcal{CA}$. More formally, the terminal must accept in some session sid for partner id pid, such that (a) pid is not adversarially controlled, and (b) there is no accepting card session for honest card pid with the same sid (including also the case that party pid has not been registered with a public key). If this happens we say that the adversary wins.

Definition 2.2 (Impersonation Resistance) *We call a protocol* Π*, running for security parameter* λ*,* (t, q_e, q_h, ϵ)*-impersonation resistant if no algorithm running in time* t*, invoking* q_e *instances of* Π *and making at most* q_h *queries to the random oracle can win the above experiment with probability greater than* ϵ*. We call the value* $\Pr[\mathcal{A} \textsf{ wins}]$ *the advantage of the algorithm* $\mathcal{A}$*, and we denote the maximum over all* (t, q_e, q_h)*-bounded* $\mathcal{A}$ *by* $\mathbf{Adv}^{ir}_{\Pi}(t, q_e, q_h)$*.*

2.3 Privacy for Key Exchange

Privacy in cryptography comes in many different flavors. The OPACITY documentation does not clarify exactly which properties the protocol is aiming for. We discuss two reasonable notions.

2.3.1 Untraceability

The notion of untraceability [53, 36] requires that any adversary, able to observe honest card-to-terminal interactions and to interact with cards and terminals in a Man-in-the-Middle fashion, cannot link two key-exchange sessions of the same card (else, the owner of the card might be traced through their card). In our analysis, we consider only a very weak form of untraceability since even this is not achieved by the OPACITY protocols. As we will see, O-ZKM is trivially traceable. The reason for O-FS not achieving untraceability is mainly due to the use of renegotiation techniques (a.k.a., persistent binding) that leaks some information about the internal state of cards and terminals.

In the *weak* untraceability experiment, the adversary invokes only a single Test session, which runs an honest execution between an honest terminal and one of two cards, all selected by the adversary. More concretely, the model is the same as before except that the Test-oracle now takes as input three identities: those of an honest terminal $\mathcal{T}$ and of two honest cards $\mathcal{C}_0, \mathcal{C}_1$. The adversary may decide on the parameters input to these parties; it suffices for our impossibility result to alway use persistent binding. The oracle (which may be queried only once) then flips a bit b and runs an honest, fresh execution between the chosen terminal $\mathcal{T}$ and either the left or the right card (depending on b). The adversary tries to predict the bit b.

Definition 2.3 ((Weak) Untraceability) *We call a protocol* Π*, running for security parameter* λ*,* (t, q_e, q_h, ϵ)-untraceable *if no algorithm* $\mathcal{A}$ *running in time* t*, invoking* q_e *instances of* Π *and making at most* q_h *queries to the random oracle, can win the above experiment with probability greater than* $\frac{1}{2} + \epsilon$*. We call the value* $\left|\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}\right|$ *the advantage of the algorithm* $\mathcal{A}$*, and we denote the maximum over all* (t, q_e, q_h)*-bounded* $\mathcal{A}$ *by* $\mathbf{Adv}^{trace}_{\Pi}(t, q_e, q_h)$*.*

For plain untraceability consider the same experiment as above, but with an adversary that can run multiple Test-sessions. Thus, we consider (t, q_e, q_t, q_h)-bounded adversaries, where q_t denotes the number of test-sessions.

2.3.2 Identity Hiding

Intuitively, an adversary against untraceability should not even link two sessions run by the same card. A weaker notion, called *identity hiding*, only stipulates that an adversary is unable to know *which* card authenticates (though she may know that she has seen this card authenticate before). Thus, untraceability hides both the identity (i.e., the certificate) of the card and its history (e.g., its state). By contrast, identity hiding only hides the certificate.

An approach previously used in the literature, e.g. [62], is to consider a distinguishing game where an adversary must distinguish between it interacting with a real card and terminal from interacting with a *simulator*, which does not know the certificates. The idea is that the simulator should be able to provide the adversary with a view of the protocol that is indistinguishable from the real one. However, the use of persistent binding makes this approach problematic. In particular, if the simulator tries to authenticate to an honest terminal, it will be rejected (as it cannot forge a valid certificate), enabling the adversary to trivially tell it apart from an honest card if it learns the outcome of the session. Moreover, even if we restrict the adversary's view such that she cannot see the result of key-exchange sessions, she can still notice, by looking at

the control bytes of terminals and cards, whether they are synchronized or not (i.e. whether they used persistent binding or not). However, let us stress the fact that, though linkability through the use of persistent binding is a valid attack against untraceability, it should *not*, intuitively speaking, be valid against identity-hiding as by this the adversary does not learn anything "real" about the identity of a card.

We choose a different approach to model identity hiding again starting from the previously defined BR-like approach for key secrecy. That is, we use the identical security model as for key exchange, but with one exception: we assume a special card $\mathcal{C}^*$ exists, for which two certified key-pairs $(\mathsf{sk}_0^*, \mathsf{pk}_0^*, \mathrm{cert}_0^*)$, $(\mathsf{sk}_1^*, \mathsf{pk}_1^*, \mathrm{cert}_1^*)$ are generated under (potentially different) identities. The adversary is initially given the certificates and public keys of all honest parties, except for $\mathcal{C}^*$, together with the assignment of the keys and certificates to the cards. The adversary also receives the two pairs $(\mathsf{pk}_0^*, \mathrm{cert}_0^*)$, $(\mathsf{pk}_1^*, \mathrm{cert}_1^*)$ and the corresponding identities (but not the information which pair is actually used). At the start of the game, a bit b is flipped and $\mathcal{C}^*$ is instantiated with $(\mathsf{sk}_b^*, \mathsf{pk}_b^*, \mathrm{cert}_b^*)$. When the Test oracle is queried, it returns the handle for card $\mathcal{C}^*$, allowing the adversary to access this card by using all the previous oracles, apart from $\mathsf{Corrupt}$. The adversary must predict the bit b, i.e. it must learn whether card $\mathcal{C}^*$ is associated with the left or right key pair. The only restriction is that the partner id pid output in any of the possibly multiple Test sessions by the tested card $\mathcal{C}^*$ is always an identity of an honest terminal (if the terminal is malicious the adversary trivially decrypts the encrypted certificate). Furthermore, no $\mathsf{Corrupt}$ queries must be issued to terminals or to $\mathcal{C}^*$.

Note that in our model the adversary does not choose the key pairs/certificates adaptively (after having received a list of valid certificates). One can easily extend the model to enable adaptive selection of the targeted card and its two potential certificates. Yet, such a model is equivalent (up to a factor equal to the square of the number of certificates) to our simplified model if key generation and certification are stateless; we can simply predict the chosen certificates with the claimed probability.

Definition 2.4 (Identity Hiding) *We call a protocol* Π*, running for security parameter* λ*,* $(t, q_e, q_t, q_h, \epsilon)$-identity-hiding *if no algorithm* $\mathcal{A}$ *running in time* t*, invoking* q_e *instances of* Π*, including* q_t Test*-sessions, and making at most* q_h *queries to the random oracle, can win the above experiment with probability greater than* $\frac{1}{2} + \epsilon$*. We call the value* $\left|\Pr[\mathcal{A}\ \mathsf{wins}] - \frac{1}{2}\right|$ *the* advantage *of the algorithm* $\mathcal{A}$*, and we denote the maximum over all* (t, q_e, q_t, q_h)*-bounded* $\mathcal{A}$ *by* $\mathbf{Adv}_{\Pi}^{id\text{-}hide}(t, q_e, q_t, q_h)$.

2.4 Deniability for Key Exchange

Deniability allows a party to deny its participation in a protocol execution when its partner likes to prove this fact to a third party. We discuss various notions of deniability [25, 23, 18, 64, 63, 22, 24].

The notion of *deniable authentication*, introduced in [25] and later on extensively studied, provides both the seemingly opposite features of authentication and deniability: the sender can prove that a given message actually originates from him, but no malicious receiver can convince anyone else that the message was sent by that sender. Formally, the communication is required to be efficiently simulatable or, in other terms, the transcripts do not reveal any evidence of the interaction. From this perspective, deniability appears very close to the notion of zero knowledge [29]. An excellent comparison which emphasizes connections and separations between the two notions can be found in [54].

Deniability has been modified for several scenarios. A weaker property called *partial deniability* [23] holds when the transcripts are only peer-independent. Consider a KE session between two parties $\mathcal{T}$ and $\mathcal{C}$ (initiator and responder). If the two parties $\mathcal{T}$ and $\mathcal{C}$ interact, and one of

the two (say, $\mathcal{C}$ is the adversary here) tries to prove, right after the interaction, that it was her interacting with the other party ($\mathcal{T}$). In other words, partial deniability prevents $\mathcal{C}$ to prove that she has been $\mathcal{T}$'s peer, but does not prevent her to show that $\mathcal{T}$ actually took part in some KE execution. Partial deniability is weaker than the standard property in that a party executing a KE session cannot later deny having done so, but still no malicious partner can provide evidence to be the actual peer. A property related to partial deniability is the so-called *peer-deniability* [18].Stronger notions of deniability have been defined, too. *Forward deniability* [22] requires the protocol to be statistical zero-knowledge. Another strong (and somewhat less intuitive) flavor of deniability, involving an *on-line* judge which can interact with protocol participants during the execution, has been introduced in [24].

Deniability in the context of KE [23] allows each participant to deny having shared a session key with its partner: after the execution, no one (neither the parties involved nor outsiders) can convince a third party that the protocol execution took place. Indeed, whatever is generated during an honest interaction could also have been produced efficiently by the adversary herself. As pointed out in [23], deniability for KE requires not only the entire transcripts, but also the session keys, to be simulatable.

Finally, let us remark that deniability and untraceability (and also identity hiding) are incomparable. Deniability covers the issue *that* an execution took place. In contrast, untraceability, and in a sense also identity hiding, disguises *who* participated, but it may still not be possible to generate the communication without this partner.

Our model of Deniable KE. For OPACITY we target only a very basic level of deniability that we call *outsider deniability*. Intuitively *outsider deniability* protects against the external abuse of transcripts between honest parties. As we will see, O-ZKM does not even achieve this simple level of protection (even if we do restrict it to not use persistent binding). For O-FS the picture is not as clear. We can show that with the use of persistence binding our model, and in fact, any of the above mentioned is not achieved. If we, however, remove persistent binding from the protocol we can show that O-FS is outsider deniable. We have not investigated whether or not stronger forms deniability are met by O-FS without persistent binding.

Outsider deniability forbids any outsider, which observes the communication between honest terminals and cards, to convince a third party (i.e., a judge) that a given card $\mathcal{C}$ and a given terminal $\mathcal{T}$ have been involved in a KE session. Note that the third party is not given the long-term secrets of the alleged card and terminal, that is, it is only given publicly available data. It is instructive to imagine that two parties $\mathcal{T}$ and $\mathcal{C}$ execute a KE session in the presence of an adversary $\mathcal{E}$ which merely eavesdrops on the communication. Informally, no message that the terminal and the card exchange should give $\mathcal{E}$ any help in proving to the judge that $\mathcal{T}$ and $\mathcal{C}$ have ever interacted. Stated differently, $\mathcal{E}$ could produce transcripts of the communication (and session keys) that 'look authentic' herself, even without observing the actual execution. As for zero knowledge, we formalize that $\mathcal{E}$ could produce 'genuine-looking' messages (and keys) by letting an efficient simulator $\mathcal{S}$ succeed with the same probability, and require that no efficient distinguisher $\mathcal{A}$ can tell, by looking at either honest transcripts or simulated ones, whether they are real or not. In our case, since we only consider honest executions, we can simply substitute $\mathcal{E}$ by the transcript of the execution.

The attack model is again identical to that of key secrecy, but for a modified Test-oracle. A query to Test contains a pair of identities $\mathcal{C}$ and $\mathcal{T}$. We assume the existence of a PPT algorithm $\mathcal{S}$ which simulates protocol runs between honest parties. In particular, the simulator receives public inputs of these parties (that is, the certificates and identities, but not the corresponding secret keys) as well as the parameters passed to the parties, and it needs to emulate the transcript as well as the the session key. Whenever the adversary invokes Test, depending

on a random bit b, either a real protocol execution is performed (i.e., an Execute-query) or $\mathcal{S}$ simulates the communication. The resulting transcripts are given to $\mathcal{A}$. Note that this means that $\mathcal{A}$ is passive during the test. The adversary's goal is to return a prediction b' of bit b.

We say that a protocol is (outsider) deniable if $\mathcal{A}$ wins in the experiment above with probability not significantly higher than 1/2. In the definition below, the adversary plays the role of a judge who receives transcripts and has to decide whether they come from a real execution or they are produced by the simulator.

Definition 2.5 (Outsider Deniability) *We call a protocol Π, running for security parameter λ, (t, q_e, q_t, ϵ)-outsider-deniable if no algorithm running in time t and invoking q_e instances of Π, including q_t Test-sessions, can win the above experiment with probability greater than $\frac{1}{2} + \epsilon$. We call the value $\left|\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}\right|$ the advantage of the algorithm $\mathcal{A}$, and we denote the maximum over all (t, q_e, q_t)-bounded $\mathcal{A}$ by $\mathbf{Adv}_{\Pi}^{deny}(t, q_e, q_t)$.*

Asymptotically, the protocol is *outsider-deniable* if every polynomial (in λ) time algorithm has advantage negligible in λ.

3 The OPACITY Protocols

The OPACITY suite consists of two key-exchange protocols, one called *OPACITY with Zero-Key Management* (O-ZKM), the other one called *OPACITY with Full Secrecy* (O-FS). In the authors' words O-ZKM is "a lightweight option best suited to protect contact or contactless transactions when terminals are not capable of protecting static secrets or supporting a hardware security module" [60]. O-FS, on the other hand is, again in the words of the authors':

> "optimized for contactless authentication transactions between a Secure Element and a remote entity, when mutual authentication is necessary or when it is necessary that the identity of the Secure Element or card holder is never revealed to unauthorized parties. An external third-party without privilege should not be able to associate the identity of the card holder to a transaction made with the card. Under this mode the protocol protects end-to-end sensitive information that needs to be transported to or from the Secure Element, and which remains sensitive and valuable after the transaction or the session is completed. For instance in key management use cases, when the key material that is communicated must remain protected for confidentiality at least during the life time of the key."[60]

Both protocols allow a terminal $\mathcal{T}$ and a card $\mathcal{C}$ to agree upon session keys $\mathsf{sk}_{\mathsf{MAC}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{RMAC}}$ (for command authentication, encryption, and response authentication). Note, however, that though subsumed under the same protocol suite, the two protocols are nonetheless quite different, the main difference being that O-ZKM has only one-sided authentication, i.e., the card authenticates to the terminal but not vice versa. In Figure 1 we give a slightly simplified description of O-FS (i.e., without persistent binding). The full protocols, as well as a line by line description are given in the appendix on pages 41.

3.1 Protocol Descriptions

Both protocols (O-ZKM and -FS) consist of two rounds, the first one initialized by the terminal. Our description closely follows the original formulation in the standards. We make, however, minor changes in notation so as to simplify the diagram and improve legibility. We also change some variable names to be more compliant to standard cryptographic descriptions of protocols. A list of variable names, their intended meanings, and a pointer to the original name is given in

Table 2 on page 16. We give a shortened description of the O-FS protocol, without renegotiation, in Figure 1.

From a bird's-eye view the O-FS protocol works as follows. Both the terminal and the card hold a certified key pair $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ and $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}})$, respectively. The protocol works over a suitable elliptic curve $\mathscr{E}$; as such, secret keys are the discrete logarithms of the corresponding public keys (for some generator G). Both parties also generate an ephemeral key pair for each session, denoted by $(\mathsf{epk}_{\mathcal{T}}, \mathsf{esk}_{\mathcal{T}})$ and $(\mathsf{epk}_{\mathcal{C}}, \mathsf{esk}_{\mathcal{C}})$. The terminal first transmits its public keys $\mathsf{pk}_{\mathcal{T}}$ (encapsulated in the certificate) and $\mathsf{epk}_{\mathcal{T}}$, together with a control byte $\mathtt{CB}_{\mathcal{T}}$ for specifying different modes and for indicating a renegotiation request. The first Diffie-Hellman key is computed via the static key $\mathsf{pk}_{\mathcal{T}}$ of the terminal and the card's ephemeral key. Analogously, the second Diffie-Hellman key is derived from the terminal's ephemeral key $\mathsf{epk}_{\mathcal{T}}$ and the card's long-term key $\mathsf{pk}_{\mathcal{C}}$. Both keys are then used in a cascade of two key-derivation steps to derive the session keys. The card replies with its encrypted certificate (for privacy reasons), a MAC for authentication, a control byte for renegotiation, and its ephemeral public key. Assuming both parties are honest, the terminal can decrypt and validate the card's certificate, validate the MAC, and compute the session keys, too.

OPACITY-ZKM. The main difference between the ZKM and the FS protocol is that in ZKM the terminal does not hold a long-term secret. Consequently, during key exchange, only the card authenticates. To protect the privacy of the card holder the protocol provides for a mode where the user-specific data from the certificate — the so called GUID— is only sent encrypted. Whether or not this mode is to be used is specified by the terminal using a bit in its control byte $\mathtt{CB}_{\mathcal{T}}(\mathtt{RET\_GUID})$. In case `RET_GUID` is not set the standard is regrettably ambiguous. We interpret the standard in the following way. In case `RET_GUID` is not set, the client sends its certificate in the clear. That is, in line 8 of the protocol description the unchanged certificate is stored in variable blindID instead of the blinded certificate. This must be the case as in line 45 the terminal extracts GUID from the certificate in case `RET_GUID` is not set. The terminal needs to know the full certificate in order to validate it.

3.2 Preliminaries

Certificates. OPACITY uses certificates in the card verifiable certificate format (CVC) which is standardized as part of ISO 7816 — Part 8 [32]. Apart from the owner's public key and an identifier for the certification authority, certificates contain application-specific data which can be used to identify the card holder. In OPACITY, this 128-bit field is called GUID and identifies the holder of the card. O-ZKM encrypts GUID using AES and the derived session key. O-FS, on the other hand, encrypts the entire certificate under an intermediate key. The (outdated) source code uses AES in CBC mode with the constant 0-vector as initialization vector. In O-FS since the key is derived freshly upon every invocation and only used for a single encryption, this should not pose a security threat. For O-ZKM, on the other hand, the session key is used; this might compromise security.

Formally we consider a certification scheme as a signature scheme where the signer is a certification authority ($\mathcal{CA}$). We denote such a scheme as a tuple $(\mathsf{C.KGen}, \mathsf{C.Sign}, \mathsf{C.Vrf})$. The key generation algorithm, executed by a $\mathcal{CA}$, outputs a pair of keys $(\mathsf{sk}_{\mathcal{CA}}, \mathsf{pk}_{\mathcal{CA}}) \leftarrow \mathsf{C.KGen}(1^{\lambda})$; given the identifier $\mathsf{ID}_{\mathcal{U}}$ of a user, plus other information relative to $\mathcal{U}$ (to specify: For sure, a static public key $\mathsf{pk}_{\mathcal{U}}$), the $\mathcal{CA}$ embeds $\mathcal{U}$'s credentials into a certificate $\mathrm{cert}_{\mathcal{U}} \leftarrow \mathsf{C.Sign}(\mathsf{ID}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}}, \ldots; \mathsf{sk}_{\mathcal{CA}})$. Everybody can validate the certificate $\mathrm{cert}_{\mathcal{U}}$ by checking whether the algorithm $\mathsf{C.Vrf}(cert_{\mathcal{U}}; \mathsf{pk}_{\mathcal{CA}})$ outputs 1.

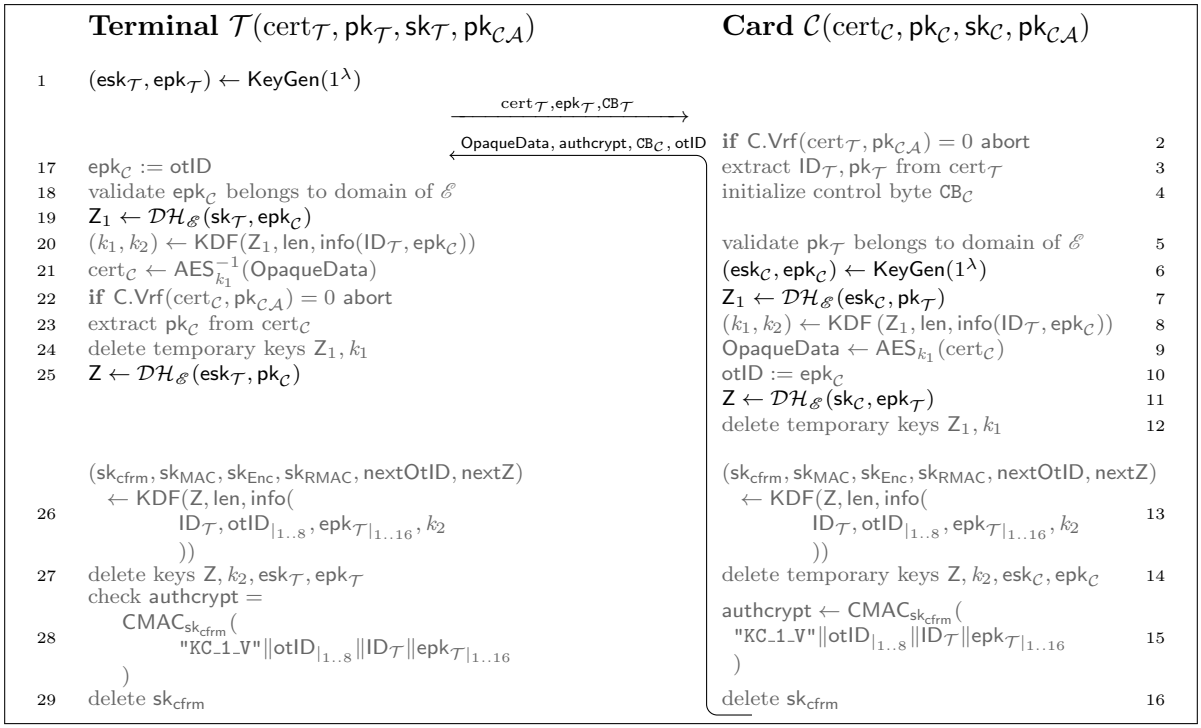

Figure 1: The shaded parts describe OPACITY with Full Secrecy without persistent binding. The complete protocol, as well as a line by line description is provided in the appendix on pages 41ff. The unshaded lines should give a high-level overview of the underlying Diffie-Hellman key exchange.

Other functionalities used by protocols. The protocols use a key-derivation function KDF as specified in NIST SP 800-56A (§5.8.1) [50], CMAC for message authentication as specified in NIST SP 800-38B [49] (CMAC is also used as PRF in the key-derivation function) and AES-128 (no mode specified). As hash function, SHA-256 or SHA-512 are deployed. In the analysis below we model KDF through a random oracle. The injective function info is defined according to NIST SP 800-56A and prepares the input to the key-derivation function (it can be thought of the length-encoded concatenation of its input). The input to info, and therefore to the key-derivation function, contains the terminal's identity $\mathsf{ID}_{\mathcal{T}}$ (not specified in detail, but we assume that this value is globally unique and also determines the terminal's certificate $\mathsf{cert}_{\mathcal{T}}$ uniquely) and usually parts of the ephemeral keys $\mathsf{otID} = \mathsf{epk}_{\mathcal{C}}$ and $\mathsf{epk}_{\mathcal{T}}$, like the leftmost 8 or 16 bytes, $\mathsf{otID}|_{1..8}$ and $\mathsf{epk}_{\mathcal{T}}|_{1..16}$, respectively.

Security Parameter. OPACITY specifies 6 parameter sets describing the length of keys and nonces, block-ciphers, and hash functions. The standard set CS2 recommends to use SHA-256 as hash function, AES-128 for encryption and MACs, and ECDH-256 for static and ephemeral keys. Nonces are 16 bytes long. By contrast, the "very strong security" setting (CS6) uses SHA-512, AES-256, ECDH-512, and 32-byte nonces. In the first case it is claimed that the resulting channel strength is 128 bits, and for CS6 the channel strength is supposedly 256.

Persistent Binding. Both protocols can be run in a renegotiation mode which gives a slight performance increase if card and terminal have already successfully exchanged keys. This mode, called *persistent binding*, requires both parties to store an intermediate secret value which is used to compute the new session keys upon renegotiation. For the O-ZKM this saves both sides a Diffie–Hellman (DH) function evaluation, and for the FS-protocol both sides save two DH-

evaluations and one AES evaluation. To indicate if persistent binding should be (or has been) used, the terminal (and resp. the card) sends a control byte $\mathtt{CB}_{\mathcal{T}}$ (resp. $\mathtt{CB}_{\mathcal{C}}$). We use the following syntax for persistent binding (PB): The variable $\mathtt{CB}_{\mathcal{T}}(\mathtt{PB})$ returns a boolean value indicating if the terminal would like to use persistent binding, if possible. The variable $\mathtt{CB}_{\mathcal{C}}(\mathtt{PB})$ is set by the card if it has used persistent binding for the current protocol execution. The variable $\mathcal{C}$.supports(PB) returns a boolean value, indicating whether the card supports persistent binding or not. In case the terminal does not support PB then it sets $\mathtt{CB}(\mathtt{PB}) = \perp$. To implement persistent binding both parties have to store certain values in a registry. We denote by PB.contains(v) a boolean value indicating whether the registry contains an entry under key v or not.

Persistent Binding in O-ZKM. An interesting difference between the two protocols is the way persistent binding (the renegotiation) is implemented. In the ZKM-protocol the card's id, which is defined as the first 8 byte of a hash of the blinded certificate is used to identify the card for persistent binding. In contrast, the FS-protocol uses a one-time identifier otID which is generated for the next session together with the current session keys. Using the card's id has the disadvantage of making the protocol trivially linkable, as the identity is sent in the clear.

3.3 Related DH Key-Agreement Protocols

We only discuss Diffie-Hellman-based key exchange protocols which are very similar in structure to OPACITY, i.e., pairwise mix static and ephemeral Diffie-Hellman keys of the partners; other protocols like (H)MQV [43, 38] or OAKE [63] are structurally further away, despite being also based on the DH protocol. The following discussion about the difference between OPACITY and closely related protocols, in particular, also applies to such more distinct protocols.

The underlying Diffie-Hellman key-agreement protocol in OPACITY resembles the structure of the Key Exchange Algorithm (KEA) [47], designed by the NSA for the FORTEZZA cryptographic suite. KEA involves two parties A and B holding a static key pair (a, aG) and (b, bG) respectively, aG and bG being public. The protocol is roughly as follows: A generates an ephemeral secret key x, computes the corresponding ephemeral public key xG and sends it to B which, in turn, generates an ephemeral key pair (y, yG) and returns yG to A. Hence, both parties compute ayG, bxG and, eventually, invoke a key-derivation function on input (ayG, bxG) to derive the session keys. A complete description of the KEA protocol, combined with a security analysis, is presented in [42]. The authors show a weakness of the scheme and suggest how to fix it by proposing a new variant KEA+, essentially a modification of KEA which includes the participants' identities in the computation of the session keys. A further modification, called KEA+C, integrates a key-confirmation phase into the protocol. A protocol very similar to KEA+ has been analyzed cryptographically in [39].

Another closely related approach are the schemes described by ANSI X9.63 [4], called "Unified Model" (UM) key-agreement protocols. The UM protocols describe several variants to combine ephemeral and static Diffie-Hellman keys for key exchange. The UM protocols have been analyzed cryptographically in [44]. Neither of the two OPACITY variants seems to be subsumable under any member of the UM family, though.

Although sharing a similar skeleton —a DH key-agreement protocol using both static and ephemeral keys— the analyses of KEA, UM and their variants [42, 39, 44] can only serve as a very vague starting point for OPACITY; the protocols differ in numerous security-relevant details. One distinctive property of our analysis here is also that we investigate low-level details more explicitly. Considering such details makes the evaluation more delicate and complex but, on the other hand, gives a more concrete perception of the (in)security of the actual protocol. This, in particular, also concerns the renegotiation step in OPACITY which is neither supported by

KEA nor by UM. Our analysis for OPACITY also needs to take additional privacy properties into account. Hence, even if OPACITY resembles the other schemes, the existing analyses provide rather weak implications for OPACITY's overall security (if any at all).

Notation	OPACITY	Format	Description
KeyGen			Generate ephemeral ECC key pair according to SP800-56A.
$\mathsf{ID}_{\mathcal{T}}$	ID_{sH}	8 byte	Identifier of terminal (part of $\mathrm{cert}_{\mathcal{T}}$ in FS).
$\mathsf{ID}_{\mathcal{C}}$	ID_{sICC}	8 byte	First 8 bytes of hash of blinded card certificate (only used in ZKM).
$\mathrm{cert}_{\mathcal{T}}$	C_H	CVC Format	Terminal's certificate in the CVC format (ISO 7816 — Part 8).
$\mathrm{cert}_{\mathcal{C}}$	C_{ICC}	CVC Format	Card's certificate in the CVC format (ISO 7816 — Part 8).
$\mathrm{cert}^*_{\mathcal{C}}$	C^*_{ICC}		Blinded card's certificate (with GUID removed).
$\mathsf{pk}_{\mathcal{CA}}$	Q_{rootH}		Verification key issued by the $\mathcal{CA}$ (to validate certificates).
$\mathsf{sk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}$	$d_{\mathrm{sH}}, Q_{\mathrm{sH}}$	SP800-56A	Terminal's *static* secret and public ECC keys.
$\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{C}}$	$d_{\mathrm{sICC}}, Q_{\mathrm{sICC}}$	SP800-56A	Card's *static* secret and public ECC keys.
$\mathsf{esk}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}$	$d_{\mathrm{eH}}, Q_{\mathrm{eH}}$	SP800-56A	Terminal's *ephemeral* secret and public ECC keys
$\mathsf{esk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}$	$d_{\mathrm{eICC}}, Q_{\mathrm{eICC}}$	SP800-56A	Card's *ephemeral* secret and public ECC keys
OpaqueData	$\mathsf{OpaqueData}_{\mathrm{ICC}}$		AES encryption of certificate or nonce $\mathrm{n}_{\mathcal{C}}$
blindID	$\mathrm{icc}_{\mathrm{ID}}$ in ZKM		Blinded certificate ($\mathrm{cert}^*_{\mathcal{C}}$) or card identifier ($\mathsf{ID}_{\mathcal{C}}$).
$\mathrm{n}_{\mathcal{C}}$	$\mathrm{N}_{\mathrm{ICC}}$	16 byte	Nonce
len	len	1 byte	Length of key material to be generated by KDF as specified in SP800-56A.
otID	$\mathrm{OTID}_{\mathrm{ICC}}$	8 byte / $\mathsf{epk}_{\mathcal{C}}$	One time identifier for persistent binding, or public ephemeral key of card.
$\mathtt{CB}_{\mathcal{T}}$	$\mathtt{CB}_{\mathrm{H}}$	1 byte	Control byte sent by terminal
$\mathtt{CB}_{\mathcal{C}}$	$\mathtt{CB}_{\mathrm{ICC}}$	1 byte	Control byte sent by card
authcrypt	$\mathrm{AuthCryptogram}_{\mathrm{ICC}}$	16 byte	Message Authentication Code, AES-128-CMAC, as specified in SP800-38B
"KC_1_V"	"KC_1_V"	6 byte	Predefined message string, value is set to "00 00 00 00 00 00"
k_1, k_2	K_1, K_2	16 byte	Temporary keys
Z_1	Z_1		Temporary secret (point on elliptic curve $\mathscr{E}$) computed during the FS protocol.
Z	Z		Shared secret (point on elliptic curve $\mathscr{E}$) used in KDF to generate session keys.
$\mathcal{DH}_{\mathscr{E}}$	EC_DH		Elliptic curve Diffie–Hellman function (relative to curve $\mathscr{E}$) specified in SP800-56A

Table 2: Notation used in protocol descriptions (lengths are for Cipher Suite CS2)

4 Security Analysis of O-FS

In the following section we present a security analysis for O-FS. A similar analysis for the weaker O-ZKM is given in Appendix A.

The concrete security parameters proposed for O-FS can be found in Section 3; however, for the sake of generality, our analysis features abstract parameters, e.g. instead of the concrete bit size of the proposed curve $\mathscr{E}$, defined on the field $\mathcal{K}$, we write $\#\mathscr{E}(\mathcal{K})$ (this is, in fact, the size of a prime-order subgroup of points). Thus, our analysis formally bounds the success probability of adversaries for *any* proposed set of parameters. We also denote the set of nonces $n_{\mathcal{C}}$ chosen by the chip card by $\mathcal{N}$, and its size by $|\mathcal{N}|$, respectively.

We note that the protocol itself is not perfectly correct in the sense that two honest parties may not derive the same session keys. The reason is that the random values otID, used as entries in the registry to identify previous connections, may yield collisions in the persistent binding registry which would disallow the parties to reconnect. However, the likelihood of this event is in the order of $q_e^2 \cdot 2^{-\ell_{\mathsf{otID}}}$ for q_e executions and ℓ_{otID}-bit values for otID. Hence, for $\ell_{\mathsf{otID}} \geq 128$, as recommended, we may simply neglect such mismatches in our analysis. Nonetheless, it would be preferable to specify the behavior for this case clearly in the protocol description.

4.1 Security Assumptions

We prove O-FS secure under the elliptic curve *Gap Diffie–Hellman* (GDH) assumption [51] (by default we assume *all* hard problems are on elliptic curves, omitting to state this explicitly). Informally, the GDH assumption states that the CDH problem remains hard even when given access to an oracle $\mathsf{DDH}(\cdot,\cdot,\cdot)$, which tells whether three group elements form a Diffie–Hellman tuple or not. More formally, let $\langle G\rangle$ be an (additive) group of prime order q and generator $G \in \mathscr{E}$. The GDH problem is (t, Q, ϵ)-hard in $\langle G\rangle$ if any algorithm $\mathcal{A}$ running in time t and making at most Q queries to DDH can, on input $\langle G\rangle, G, sG, tG$, for random s, t, compute stG with probability at most ϵ. We write $\mathbf{Adv}^{\mathrm{GDH}}(t, Q)$ for (a bound on) the probability of any (t, Q)-bounded $\mathcal{A}$ solving the GDH problem.

Definition 4.1 (GDH Assumption.) *The Gap Diffie–Hellman problem is (t, Q, ϵ)-hard, in a group $\langle G\rangle$ of prime order q and generator G, if any algorithm $\mathcal{A}$ with running time t, which makes at most Q queries to a DDH oracle, wins the following experiment with probability at most ϵ:*

pick $x, y \in \mathbb{Z}_q$
compute $Z \leftarrow \mathcal{A}^{DDH(\cdot,\cdot,\cdot)}(\langle G\rangle, G, xG, yG)$
output 1 *iff* $Z = \mathcal{DH}(xG, yG)$

We use standard cryptographic notation for the other involved primitives. The certification scheme $\mathsf{Cert} = (\mathsf{C.KGen}, \mathsf{C.Sign}, \mathsf{C.Vrf})$ is modeled as a signature scheme where the signer is a certification authority ($\mathcal{CA}$); $\mathbf{Adv}^{\mathrm{forge}}_{\mathsf{Cert}}(t, Q)$ denotes the maximal probability of forging a fresh certificate within t steps and after requesting at most Q certificates. We use $\mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{AES}}(t, Q)$ to denote the maximal probability of distinguishing AES ciphertexts (in CBC mode) within t steps for at most Q challenge ciphertexts (see the remark in Section 3.2 about the actual encryption mode), and $\mathbf{Adv}^{\mathrm{forge}}_{\mathsf{CMAC}}(t, Q)$ for the maximal probability of forging a CMAC in t steps after seeing at most Q MACs. Finally, the key-derivation function (KDF) is modeled as a random oracle.

4.2 Key Secrecy

For the key-secrecy proof we consider sessions as indicated in Section 2, such that the session id sid for O-FS is set as $\mathsf{sid} = (\mathsf{otID}|_{1..8}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}|_{1..16})$; the partner id pid is set to the identity $\mathsf{ID}_{\mathcal{T}}$ on the card's side resp. to the unique card's identity GUID on the terminal's side. We observe that session id's are usually preferred to comprise the entire communication transcript. The reason is that, roughly, the more information contained in sid, the "tighter" the binding of session keys to specific executions. In this sense, our formally more loose (but, according to the protocol, presumably inevitable) choice for sid's here ties executions to partners, identified via parts of the public keys and the ephemeral keys. Indeed, one easy enhancement for the protocol would be to include the card's certificate in the key-derivation step, or at least its entire public key.

The next theorem shows that O-FS is secure as a key agreement protocol, i.e., O-FS provides key secrecy. We stress that the theorem and its proof cover the full protocol, including renegotiation.

Theorem 4.2 (Key Secrecy of O-FS) *In the random-oracle model, we have*

$$\mathbf{Adv}^{ake}_{\Pi_{\mathsf{OFS}}}(t, q_e, q_h) \leq \mathbf{Adv}^{forge}_{\mathsf{Cert}}(t, q_e) + \frac{3q_e(2q_e + q_h)}{2^{\min\{\ell_{k_2}, \ell_{\mathsf{Z}}\}}} + 2q_e^2 \cdot \mathbf{Adv}^{\mathrm{GDH}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h)$$

where λ denotes the security parameter, t the running time of adversary $\mathcal{A}$, and q_e (resp. q_h) the number of executions (resp. queries to the random oracle), and ℓ_{k_2} and ℓ_{Z} denote the bit lengths of values k_2 resp. Z.

Proof. The approach we use to prove security is game-based: we start from GAME_0, the original attack against key secrecy, and gradually turn it into GAME_6 via intermediate experiments in such a way that the probability of winning GAME_i and GAME_{i+1} are negligibly close, and, in last game, the adversary wins with target probability 1/2 exactly.

Description of GAME_0. This corresponds to the original attack on the O-FS protocol Π_{OFS}.

Description of GAME_1. This game works as GAME_0 but aborts if the adversary successfully sends to an honest party a valid certificate for a public key that has not been registered with the certification authority $\mathcal{CA}$ before. In particular, this would mean that the adversary must have forged a certificate for a chosen identity and public key. In that case, this adversary can be straightforwardly turned into a successful attacker against the underlying certification scheme, i.e., she outputs a "fresh" certificate which verifies under public key $\mathsf{pk}_{\mathcal{CA}}$. This is done by simulating the key exchange game, knowing all the parties secrets, and playing against the CA. Thus, we have

$$\Pr[\text{GAME}_0] \leq \Pr[\text{GAME}_1] + \mathbf{Adv}^{\text{forge}}_{\mathsf{Cert}}(t, q_e).$$

We next show that the adversary most likely will not make a call to the KDF in the first invocation (line 8) about the same input as honest cards:

Description of GAME_2. This game works as GAME_1 but now, if the adversary $\mathcal{A}$ makes a hash query to the KDF for a Diffie–Hellman key Z_1 computed by an honest card for a session without persistent binding in which it receives the identity $\mathsf{ID}_{\mathcal{T}}$ of an honest terminal in the first message, we abort the game.

More precisely, the "bad" event leading to abort happens whenever $\mathcal{A}$ queries the KDF on $(\mathsf{Z}_1 = \mathcal{DH}(\mathsf{epk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{C}}), *, \mathsf{info}(*, \mathsf{epk}_{\mathcal{C}}))$ such that there exist an accepting session where $\text{cert}_{\mathcal{T}}$ with its unique identity $\mathsf{ID}_{\mathcal{T}}$ points to an honest party. We now show that the probability of querying the KDF on this value is bounded by the probability of breaking the GDH assumption. Suppose $\mathcal{A}$ queries the KDF for a DH key as defined above: we can turn $\mathcal{A}$ into an adversary $\mathcal{A}_{\text{GDH}}$ which breaks the GDH assumption. Recall that given (the description of the group and) two group elements sG, tG, the task of $\mathcal{A}_{\text{GDH}}$ is to compute stG. Informally, $\mathcal{A}_{\text{GDH}}$ uses the following technique. She tries to guess the session $\mathcal{C}_i$ (and a partner session $\mathcal{T}_j$) for $i, j \in \{1, \ldots, q_e\}$ for which the adversary $\mathcal{A}$ queries on the DH key $\mathcal{DH}(\mathsf{esk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{T}})$. For this card session $\mathcal{C}_i$, we set the ephemeral public key as $\mathsf{epk}_{\mathcal{C}} = sG$, and the long-term public key of the terminal as $\mathsf{pk}_{\mathcal{T}} = tG$. Here we use the fact that the card session involves some $\text{cert}_{\mathcal{T}}$ received in the first message, which uniquely identifies the honest terminal, its certificate, and its public key $\mathsf{pk}_{\mathcal{T}}$ (according to GAME_1). In other words we can safely inject the given public key. In fact, for an adversarially controlled terminal the adversary would otherwise be able to compute the key Z_1.

If the guess of $\mathcal{A}_{\text{GDH}}$ is correct, at some point $\mathcal{A}$ asks the right query $(\mathsf{Z}_1, *, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}))$: $\mathcal{A}_{\text{GDH}}$ can now invoke the DDH oracle to detect this query, i.e., $\mathcal{A}_{\text{GDH}}$ can check whether $\text{DDH}(sG, tG, \mathsf{Z}_1) = 1$. Finally, $\mathcal{A}_{\text{GDH}}$ outputs Z_1 and stops, winning the game. We do, however, encounter one problem here: We may need to be able to compute DH keys for other sessions of the parties we have injected the keys for, i.e., if the adversary invokes a session of the terminal $\text{cert}_{\mathcal{T}}$ with a malicious card such that the adversary could in principle compute the DH key easily while $\mathcal{A}_{\text{GDH}}$ cannot. In order to overcome this we use some "list-based" approach

basically storing the DH values implicitly through (unordered) pairs $\{X, Y\}$ and use our DDH oracle to check if a query about some W is for the "right" key $W = \mathcal{DH}(X, Y)$; if so, we answer consistently, else we can assign a new random value and store the new entry in the list. Hence, we actually need two lists, one for queries $\{X, Y\}$ of honest parties, and one for queries W of the adversary, and we need to make sure that both lists are consistent all the time. If we succeed in doing so, then the approach is perfectly aligned to answer hash queries with a random oracle; it is even more fine grained.

We discuss more rigorously how $\mathcal{A}_{\mathrm{GDH}}$ simulates a suitable 'key-exchange environment' for $\mathcal{A}$, i.e., the way she answers queries requested by $\mathcal{A}$ or by parties monitored by her. Basically, $\mathcal{A}_{\mathrm{GDH}}$ simulates all steps of honest parties in $\mathcal{A}$'s attack with an important difference. We let $\mathcal{A}_{\mathrm{GDH}}$ store tuples $(\mathsf{honest}, \{S, T\}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$ for queries $(\mathcal{DH}(S, T), *, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}))$ made by honest parties, and use tuples of the form $(\mathsf{malicious}, \mathsf{Z}_1, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$ for adversarially chosen queries, where y denotes the answer. Note that we use the unordered pair $\{S, T\}$ as an implicit representation of $\mathcal{DH}(S, T) = \mathcal{DH}(T, S)$ which we cannot compute, but for which we can test equality with any group element later via the DDH oracle. The simulation in more detail works as follows.

Answers to $\mathcal{A}$'s queries to KDF. For any tuple $(\mathsf{Z}_1, *, *)$, $\mathcal{A}_{\mathrm{GDH}}$ invokes the DDH oracle on input (sG, tG, Z_1).

- If $\mathrm{DDH}(sG, tG, \mathsf{Z}_1) = 1$, then $\mathcal{A}_{\mathrm{GDH}}$ outputs Z_1 and halts. In this case $\mathcal{A}_{\mathrm{GDH}}$ has already found the DH key and can abort.
- Otherwise, $\mathcal{A}_{\mathrm{GDH}}$ checks whether the value Z_1 has already appeared in previous executions of the protocol: if there exist entries $(\mathsf{honest}, \{X, Y\}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$ such that $\mathrm{DDH}(X, Y, \mathsf{Z}_1) = 1$, or $(\mathsf{malicious}, \mathsf{Z}_1, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$, then $\mathcal{A}_{\mathrm{GDH}}$ returns y.
- In case no such entries are found, the adversary $\mathcal{A}_{\mathrm{GDH}}$ picks a random value y, stores the tuple $(\mathsf{malicious}, \mathsf{Z}_1, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$, and returns y.

Answers to honest parties' queries to KDF. Whenever an honest party P requests $(\mathsf{Z}_1, *, *)$, we have to distinguish two cases:

- An honest party P asks a query $(\mathsf{Z}_1, *, *)$, where $\mathsf{Z}_1 \leftarrow \mathcal{DH}(X, Y)$ such that P knows either $\log X$ or $\log Y$ (i.e. values x, y such that $X = xG$ and $Y = yG$). In other words, P did not receive one of the injected public (resp. ephemeral) keys sG or tG, but a complete key pair (x, xG) generated by $\mathcal{A}_{\mathrm{GDH}}$. Hence P can compute the value Z_1 itself: it does so and, eventually, makes the oracle-query. Then, adversary $\mathcal{A}_{\mathrm{GDH}}$ returns y such that there is an entry $(\mathsf{honest}, \{X, Y\}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$ where $\mathsf{Z}_1 = \mathcal{DH}(X, Y)$, or, otherwise, y is taken from a tuple $(\mathsf{malicious}, \mathsf{Z}_1, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$. If $\mathcal{A}_{\mathrm{GDH}}$ has not already stored such a tuple, a new entry $(\mathsf{honest}, \{X, Y\}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$ for a random y is created by her.
- An honest party P is supposed to derive $\mathsf{Z}_1 = \mathcal{DH}(S, T)$, but it knows neither $s = \log S$ nor $t = \log T$ (i.e., P is one of the parties which received injected public keys sG and tG). In this case, $\mathcal{A}_{\mathrm{GDH}}$ looks for entries $(\mathsf{honest}, \{S, T\}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$ in the list of honest entries; if she finds none, she looks for a malicious entry $(\mathsf{malicious}, \mathsf{Z}_1, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}, y)$ which satisfies $\mathrm{DDH}(S, T, \mathsf{Z}_1) = 1$. If she finds such an entry she can again abort prematurely with success. If there are no such entries, she picks y uniformly at random. Eventually, she returns y. If no successful query has been requested then, when $\mathcal{A}$ outputs her verdict b, then $\mathcal{A}_{\mathrm{GDH}}$ returns $\mathsf{failure}$ and stops.

Now, we have to rely on the adversary $\mathcal{A}$ making her successful query in session $\mathcal{C}_i$ with our guess of partner $\mathcal{T}_j$. This happens with probability at least $1/q_e^2$ because $\mathcal{A}$'s view is independent of the choice of i and j. In that case, $\mathcal{A}_{\mathrm{GDH}}$ successfully locates and outputs the DH value of the GDH challenge. Furthermore, the simulation is otherwise faithful and $\mathcal{A}_{\mathrm{GDH}}$ can, for example, answer Reveal queries with help of the simulated random oracle.

The running time of $\mathcal{A}_{\mathrm{GDH}}$ is $t + O(\lambda \cdot q_e \log q_e)$ due to maintaining the oracle-queries list for entries of assumed size $O(\lambda)$, and she makes at most $2q_e + q_h$ queries to the DDH oracle. Hence, we have

$$\Pr[\mathrm{GAME}_1] \leq \Pr[\mathrm{GAME}_2] + q_e^2 \cdot \mathbf{Adv}^{\mathrm{GDH}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h).$$

The next game extends the argument to the second KDF call (line 13), but now also including the case of persistent binding being used.

Description of GAME_3**.** This game works as GAME_2 but now, if the adversary $\mathcal{A}$ makes a hash query to the KDF for a Diffie–Hellman key Z and k_2 computed by an honest card, for a session in which it receives the identity $\mathsf{ID}_{\mathcal{T}}$ of an honest terminal in the first message, we abort the game.

According to the previous game, the adversary never queries the first KDF (in a session without persistent binding) about the input Z_1. Hence, from $\mathcal{A}$'s point of view the part k_2 in such an execution is basically an unknown value (noting that the k_1 part which is used to compute the ciphertext is independent). The only information the adversary has about k_2 are random oracle values derived from it, e.g., through a Reveal query to the card to obtain the session keys, but unless the adversary "accidentally" finds k_2 by querying the random oracle about the entire pre-image involving k_2 —which is even harder than merely finding k_2 — the value remains unknown to $\mathcal{A}$. Taking into account the at most $2q_e$ distinct queries of honest parties to the random oracle, conservatively assuming that $\mathcal{A}$ knows these values, the probability of finding the pre-image including the random ℓ_{k_2}-bit value k_2 is at most $(2q_e + q_h) \cdot 2^{-\ell_{k_2}}$. Summing over all at most q_e values for k_2 appearing in executions of honest cards yields that the probability of $\mathcal{A}$ querying the random oracle about some k_2 value, is at most $q_e(2q_e + q_h) \cdot 2^{-\ell_{k_2}}$.

The same line of reasoning now applies to the value nextZ in such executions, inductively for all subsequent renegotiation runs. The probability of the adversary asking the ℓ_{Z}-bit value nextZ is bounded above by $q_e(2q_e + q_h) \cdot 2^{-\ell_{\mathsf{Z}}}$. In summary we thus have

$$\Pr[\mathrm{GAME}_2] \leq \Pr[\mathrm{GAME}_3] + \frac{2q_e(2q_e + q_h)}{2^{\min\{\ell_{k_2}, \ell_{\mathsf{Z}}\}}}.$$

Next we turn to the case of honest terminals. Note that the situation is slightly different than in case of an honest card, because the adversary can in principle send an ephemeral key $\mathsf{epk}_{\mathcal{C}}$ for which it knows the secret key and can thus compute Z_1 and query the KDF in the first invocation. This has not been possible for the terminal's certified public key in the previous games; it concretely shows up in the reduction to the Gap Diffie-Hellman problem where we injected the given key as $\mathsf{pk}_{\mathcal{T}}$ into the simulation. However, we can still conclude that the input to the second key derivation step cannot be computed by the adversary.

Description of GAME_4**.** This game works as GAME_3 but now we abort the game, if the adversary $\mathcal{A}$ makes a hash query to the KDF for a Diffie–Hellman key Z computed by an honest terminal for a session without persistent binding in which it receives a valid certificate $\mathrm{cert}_{\mathcal{C}}$ for user GUID of an honest card.

The reduction to the GDH problem follows as in the case of GAME_2:

$$\Pr[\mathrm{GAME}_3] \leq \Pr[\mathrm{GAME}_4] + q_e^2 \cdot \mathbf{Adv}^{\mathrm{GDH}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h).$$

The next game also covers the case of a renegotiation step:

Description of GAME_5**.** This game works as GAME_4 but now, if the adversary $\mathcal{A}$ makes a hash query to the KDF for a value Z computed by an honest terminal for a session in which it received (or recovered from the registry) user identity GUID of an honest card, we abort the game.

If persistent binding is not used then the claim follows from the previous game. Else, it inductively follows as in GAME_3 that the adversary queries the random oracle about Z with probability at most $q_e(2q_e + q_h) \cdot 2^{-\ell_{\mathsf{Z}}}$:

$$\Pr[\text{GAME}_4] \leq \Pr[\text{GAME}_5] + \frac{q_e(2q_e + q_h)}{2^{\ell_{\mathsf{Z}}}}.$$

We have reached the point where no adversary queries the pair Z and k_2 derived in a session of an honest party with a partner identity of an honest party. We note that adversarially controlled partners for the tested session are ruled out by definition, and unregistered identities cannot occur in test sessions according to GAME_1. However, this still does not necessarily yield a secure key exchange protocol, since instead of computing the session keys, an adversary could enforce the same input to the final KDF function with identical pair Z and value k_2 in another execution with an honest party. Then, if another honest session yielded the same input, a Reveal-query for this session would clearly help the adversary to distinguish the tested session key from random. We next argue why this event cannot happen. To this end recall that such a Reveal-query must be for an unpartnered session, i.e., with a different session id sid.

Description of GAME_6**.** This game works as GAME_5, but aborts if there are two honest parties with both accepting sessions yielding identical input to the final KDF but which are not partnered.

Note that $\mathsf{sid} = (\mathsf{otID}|_{1..8}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}|_{1..16})$ and that the function info is injective. Hence, the data from session identifiers enters the second KDF evaluation (for both new and renegotiation steps) such that distinct sid's for unpartnered sessions yield distinct input to the KDF. Hence, any call to KDF by another honest unpartnered party is for a different input than in the test session. Put differently, the derived key in a tested session is independent from keys derived by other (unpartnered) parties.

$$\Pr[\text{GAME}_5] \leq \Pr[\text{GAME}_6].$$

Thus, we have eventually reached a game where the adversary can merely guess the secret bit b because no unpartnered sessions have the same KDF input and the adversary never queries the KDF function on the same input as an honest party does in a fresh session. Hence, $\Pr[\text{GAME}_6] \leq \frac{1}{2}$. Summing up all probabilities for game transitions yields the claim for the original attack in GAME_0. □

Note that key secrecy does not rely on the security of the authenticated encryption (which only enters the impersonation resistance proof), nor the secrecy of the certificate (which is only used for privacy). At the same time neither step does harm to key secrecy.

(Weak) Forward Secrecy. Next, we investigate the (weak) forward secrecy of the O-FS protocol, where we allow the Test-oracle to be queried on (w)fsec-fresh sessions. We start with weak forward secrecy. Here, party corruptions are allowed only *after* the (fresh) test session ended in an accepting state. So, even if the long-term secrets and internal states of one participant leaks, any previous session keys are still confidential.

Learning the long-term key of only one party is now no longer sufficient to break key secrecy. More precisely, terminal corruptions enables one to emulate the terminal's behavior and compute the intermediate secret Z_1. However, the value Z remains unknown and querying the random oracle on Z still reduces to the GDH problem. Analogously, learning the card's long-term secret $\mathsf{sk}_{\mathcal{C}}$ can be used to find Z, but Z_1 (and hence, k_2) still look random to the adversary.

The internal state of each party consists of the values nextZ and $\mathsf{nextOtID}$, needed in persistent binding. These values, together with the session keys for this particular execution, are obtained by calling the KDF on previously computed values Z and k_2. Since Z and resp. k_2 are computed from fresh ephemeral secrets, resp. sampled randomly, this preimage remains unknown to the adversary. Thus, she cannot recompute the hash value so as to learn the session key for the fresh instance queried to the Test-oracle. In the random oracle model, these session keys remain private.

Note that the notion wfsec-freshness only allows for the corruption of one participant—more precisely, the party whom the session belongs to—of the execution. Thus O-FS provides weak forward secrecy. Yet, we cannot prove forward secrecy for O-FS. In fact, an adversary knowing the state of a party, in particular the persistent binding table, can compute the session key by looking up the correct partner and corresponding nextZ and transcript generated by an instance through an $\mathsf{Execute}$-query. In other words, if persistent binding is used, no ephemeral or long-term secrets are used in computing the session key, but rather a persistent binding table entry. Clearly, this cannot achieve forward secrecy.

On Further Desirable Security Properties. According the resistance against Key-Compromise Impersonation (KCI) attacks we observe the following. Let us consider first the case where a terminal obtains the long-term secret of a chip card, i.e., $\mathsf{sk}_{\mathcal{C}}$. This secret is used to compute the value Z on the card's side. However, it only requires ephemeral secrets of the terminal to compute this value, and thus, it does not help the terminal to authenticate and, in particular, to compute or obtain the input k_2 to the KDF. The same arguments hold for the opposite case. The knowledge of a terminal's long-term secret $\mathsf{sk}_{\mathcal{T}}$ is used to compute Z_1 which a card can compute by its ephemeral secrets. The intermediate secret Z remains unknown to the card. The card would need to forge a MAC to authenticate and still would not be able to distinguish the session key from random.

On possession of both ephemeral secret keys $\mathsf{esk}_{\mathcal{C}}$ and $\mathsf{esk}_{\mathcal{T}}$ an adversary can compute both intermediate secrets Z_1 and Z. Hence, O-FS does not provide security against leakage of internal randomness.

4.3 Impersonation Resistance

In this section we show that O-FS achieves impersonation resistance. Recall that this means that a malicious card cannot make an honest terminal accept, unless it is a pure relay attack and there is a card session with the same sid.

Theorem 4.3 (Impersonation Resistance of O-FS) *In the random-oracle model, we have*

$$\begin{aligned}\mathbf{Adv}^{ir}_{\Pi_{\mathsf{OFS}}}(t, q_e, q_h) \;\leq\; & 2q_e \cdot \mathbf{Adv}^{forge}_{CMAC}(t + O(\lambda \cdot q_e \log q_e), 0) \\ & + 4q_e \cdot \mathbf{Adv}^{ake}_{\Pi_{\mathsf{OFS}}}(t, q_e, q_h)\end{aligned}$$

where λ denotes the security parameter, t the running time of adversary $\mathcal{A}$, and q_e (resp. q_h) the number of executions (resp. queries to the random oracle).

The proof follows (almost) directly from the key secrecy proof, noting that in order to be able to impersonate one would need to compute a MAC for the secure key $\mathsf{sk}_{\mathsf{cfrm}}$.

Proof. The proof is straightforward given the key secrecy of the scheme, together with the property that the card computes a message authentication code over the (encrypted) certificate. Assume that the adversary mounts an impersonation attack and that there exists a session sid for which the honest terminal decrypts, accepts the MAC and the certificate. Then we show how to break key secrecy as follows: Initially guess one of the at most q_e executions and when the terminal in this execution receives the card's message, then call the Test-oracle to receive either the session key or a random string. Here we slightly abuse notation and assume that our adversary also receives $\mathsf{sk}_{\mathsf{cfrm}}$ or a random key, in addition to the other key components; by symmetry to the other keys this does not violate key secrecy in this case here. Next try to decrypt the card's certificate, to verify the MAC and the certificate. If this succeeds then output $b' = 0$, else return a random bit b'.

Note that the proof of key secrecy actually shows that the terminal rejects any invalid or not issued certificate; this is already covered by the bound for key secrecy. This also holds for collisions among session keys among unpartnered sessions. Both properties are also necessary for key secrecy and thus are taken care of by the key-secrecy advantage. Also observe that the adversary against impersonation resistance does not make Test-queries, hence our adversary makes at most one Test-query. Furthermore, the terminal has not been corrupted at that point, and there is no partnered card. Hence, the terminal's session is fresh, and disclosing $\mathsf{sk}_{\mathsf{cfrm}}$ (or a random key) cannot help to break key secrecy because this key has not been used intentionally by an honest party yet; it may of course be the case that an independent input to the key derivation function yields the same output, but this is irrelevant for the analysis (because it is already covered by the MAC unforgeability that another randomly chosen key matches the attacked key).

Given that the impersonation adversary is successful with non-negligible probability, we would hence guess the "right" execution with probability $1/q_e$. In this case, we would predict b correctly with non-negligible probability: If $b = 0$ then our prediction b' is correct; if $b = 1$ then we can only accept if the MAC verifies under the independent random key output in the Test-query. The probability here is bounded by the probability of forging a MAC in a key-only attack. The latter is negligible by assumption. In any other case we guess b correctly with probability $\frac{1}{2}$.

Overall, the success probability for predicting b is given by at least

$$\begin{aligned} \tfrac{1}{2} + \tfrac{1}{2q_e} \cdot \mathbf{Adv}^{\mathrm{ir}}_{\Pi_{\mathsf{OFS}}}(t, q_e, q_h) \quad & \text{given } b = 0 \\ \tfrac{1}{2} - \mathbf{Adv}^{\mathrm{forge}}_{\mathrm{CMAC}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h), 0) \quad & \text{given } b = 1. \end{aligned}$$

Rearranging the equations this yields in the claimed advantage. Namely,

$$\begin{aligned} \mathrm{Prob}\left[b = b'\right] - \tfrac{1}{2} &= \tfrac{1}{2} \cdot \mathrm{Prob}\left[b = b' \;\middle|\; b = 0\right] + \tfrac{1}{2} \cdot \mathrm{Prob}\left[b = b' \;\middle|\; b = 1\right] - \tfrac{1}{2} \\ &\geq \tfrac{1}{4q_e} \cdot \mathbf{Adv}^{\mathrm{ir}}_{\Pi_{\mathsf{OFS}}}(t, q_e, q_h) - \tfrac{1}{2} \cdot \mathbf{Adv}^{\mathrm{forge}}_{\mathrm{CMAC}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h), 0), \end{aligned}$$

and observing that $\mathrm{Prob}[b = b'] - \frac{1}{2} \leq \mathbf{Adv}^{\mathrm{ake}}_{\Pi_{\mathsf{OFS}}}(t, q_e, q_h)$, we get the desired bound. □

4.4 Privacy

4.4.1 Untraceability

It is easy to see that the fully-fledged O-FS protocol (with renegotiation, a.k.a., persistent binding) is not untraceable: this is because the control byte reveals some information about the internal state of the card trying to authenticate. Persistent binding also allows other attacks, such as denial-of-service (DoS) or desynchronization attacks. Essentially, the adversary can

ensure that a targeted card stores persistent-binding information for a particular (honest) terminal, while the terminal does *not* store the same information; in the next honest session run between them, the parties are desynchronized and thus the card will not authenticate. Indeed, let an adversary $\mathcal{A}$ first run Execute for a card $\mathcal{C}_0$ and a terminal $\mathcal{T}$ (thus initializing persistent binding for this card in the next session). Then $\mathcal{A}$ forwards $\mathcal{T}, \mathcal{C}_0$, and another card $\mathcal{C}_1$ to the Test-oracle. Then the adversary observes the honest control byte $\mathtt{CB}_{\mathcal{C}_b}$ output by the challenge card in its execution with the honest terminal $\mathcal{T}$: if $\mathtt{CB}_{\mathcal{C}_b}$ denotes that persistent binding is used, then $\mathcal{A}$ guesses $b = 0$; else, she sets $b = 1$. In practice this attack models an adversary able to tell whether a card was identified by a specific terminal before or it is a new user.

A different attack, frequent in authentication scenarios, is a denial-of-service (DoS) attack, where the adversary first runs a Man-in-the-Middle interaction between an honest terminal $\mathcal{T}$ and an honest card $\mathcal{C}_0$. Now $\mathcal{A}$ forwards the terminal's first message to the card; eventually $\mathcal{C}_0$ generates the keys and stores the identity of $\mathcal{T}$ and the persistent binding data in its memory. Then the card sends its message to the terminal, including the control byte $\mathtt{CB}_{\mathcal{C}}$, which denotes that the card has used persistent binding. The adversary either flips this bit or simply drops the message, making the terminal reject and, in particular, *not* compute the value nextZ. Now $\mathcal{A}$ runs the Test-query for the same terminal $\mathcal{T}$, the card $\mathcal{C}_0$, and a new card $\mathcal{C}_1$. When an honest execution is run between $\mathcal{T}$ and the challenge card, if this is card $\mathcal{C}_0$, then it generates a random value of OpaqueData, which will be rejected by the terminal, whereas card $\mathcal{C}_1$ *will* be accepted by the terminal. The adversary guesses bit b accordingly.

In practice the attacks, though perfectly valid, have their limitations. In particular, an adversary can recognize a desynchronized card *only once* and only when the card approaches a terminal it has already visited. The desynchronization is not permanent, as the card and terminal can simply run their key agreement from scratch (rather than use persistent binding). Further note that if we do not consider persistent binding, then O-FS achieves untraceability (this follows essentially as for deniability, see Section 4.5).

4.4.2 Identity Hiding

Though O-FS does not attain untraceability, it does, nevertheless, provide identity hiding. This holds as long as we assume that the unspecified mode of encryption of $\mathrm{cert}_{\mathcal{C}}$ with AES is secure (see our remark in Section 3.2).

Theorem 4.4 (Identity-Hiding in O-FS) *In the random-oracle model, we have*

$$\begin{aligned}\mathbf{Adv}^{\textit{id-hide}}_{\Pi_{\mathsf{OFS}}}(t, q_e, q_t, q_h) \leq\ & \frac{1}{2} + \mathbf{Adv}^{\textit{forge}}_{\mathsf{Cert}}(t, q_e) + \frac{2q_t(2q_t + q_h)}{2^{\ell_{k_2}}} \\ & + q_e^2 \cdot \mathbf{Adv}^{\mathsf{GDH}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h) \\ & + \frac{q_e q_t}{\#\mathscr{E}(\mathcal{K})} + q_t \cdot \mathbf{Adv}^{\textit{IND-CPA}}_{\mathsf{AES}}(t + O(q_t))\ .\end{aligned}$$

where λ denotes the security parameter, t the running time of the adversary, and q_e (resp. q_t, q_h) the number of executions (resp. Test*-sessions and queries to the random oracle).*

Proof. Very roughly, we need to show that no information about the certificate of card $\mathcal{C}^*$ is leaked. One potential threat in this regard is the encrypted transmission of the certificate under key k_1. But the adversary may also be able to deduce something about the certificate from the other data sent by the card, e.g., through the authentication data which depends on the card's public key which, in turn, is linked to the certificate and its identity.

We can re-use the proof for key secrecy to a large extend, applying the same initial game hops. The essential difference, however, is that it now no longer matters if the key sessions

leak or not: indeed, the identity hiding game must hide the assignment of certificates to cards, rather than the derived session keys. From a more technical point of view, we no longer have the concept of freshness, and Reveal queries can be made at arbitrary times.

We first give a sketch of the first part of the proof re-using the steps for key secrecy. The original game GAME_0 is the identity-hiding game. In a first game hop, in GAME_1 we exclude the possibility that an adversary forges a certificate for a valid terminal (this is identical to the first game hop for the key-secrecy proof), and we lose a term equal to the advantage of a forger against the certification scheme. In the second game hop, GAME_2 is lost if the adversary queries the same input Z_1 to the random oracle used by the honest test card, in a test session without persistent binding (again, this game hop is identical as the transition to GAME_2 in the key secrecy proof). We lose a term $q_e^2 \cdot \mathbf{Adv}^{\mathsf{GDH}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h)$. At this point, we can argue that k_1 and k_2 are independent random strings.

The next game hop is again similar to the key-secrecy proof: in GAME_3 we abort the game if the adversary makes a hash query for values Z and k_2 computed by the honest test card. As in the key-secrecy proof, this part is not dependent on whether persistent binding is used, the argument running as before. There is, however, a small modification. In the key-secrecy proof, we must account for an adversary guessing the value of nextZ, in case persistent binding is used. This was a crucial step in the key-secrecy proof, as it enabled to compute session keys for persistent binding. However, in identity-hiding, it is immaterial whether the adversary can compute the session key for a future session. Indeed, if persistent binding is used, then the string OpaqueData is random, thus revealing no information about the certificate, and no further computations using the secret or public keys of the card are made in generating authcrypt (which always verifies, since the test card is honest). Thus, we only lose a factor $\frac{2q_t(2q_t+q_h)}{2^{\ell_{k_2}}}$ in the reduction.

At this point, our proof diverges from the key-secrecy proof, since our focus here is not to hide the session keys, but rather to hide the certificate of the test card. We describe the next games in more detail.

Description of GAME_4**.** This game is equivalent to the previous game, except that the honest terminals and the test card $\mathcal{C}_b^*$ use an independent and randomly chosen $\mathsf{sk}_{\mathsf{cfrm}}$ for any session in which the KDF was queried on some honest input (Z, k_2) which appears in any of the q_t test sessions. More specifically, if such a query was made, *all* honest parties making the query change the key to the *same* random confirmation key. This ensures that the test cards use random confirmation keys (not depending on the certificates), and thus the string authcrypt reveals no information about the certificate. Crucially, note that terminals *also* use the same random key for the sessions where they are partnered with the honest test card. This ensures that the terminal always verifies authcrypt for a tested card, making this game indistinguishable from GAME_3. Note that at this point, the adversary's view is inconsistent with the outputs of the random oracle; however, the adversary never queries the random oracle on inputs (Z, k_2) used in test sessions (by GAME_3). We lose no security in this game hop.

Description of GAME_5**.** This game is identical to the previous game, except that we abort and declare the adversary to lose if any honest card chooses the same value $\mathsf{epk}_{\mathcal{C}}$ as the value chosen by the test card in any of the q_t test sessions. The reason is that we would like the keys k_1 used by the cards in the test sessions to be independent: so far, we have only proved that they are unknown to the adversary. Indeed, in the next game we will use a similar technique as in GAME_4 to argue that k_1 can be changed to a value chosen uniformly and independently at random. The probability of a collision among the keys $\mathsf{epk}_{\mathcal{C}}$ appearing in one of the q_e executions and one of q_t test executions is at most $q_e q_t \cdot \frac{1}{\#\mathscr{E}(\mathcal{K})}$. If no collision occurs, then since info is

injective, all test sessions evaluate the KDF on distinct inputs $(\mathsf{Z}_1, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}))$. Thus,

$$\Pr[\text{GAME}_4] \leq \Pr[\text{GAME}_5] + \frac{q_e q_t}{\#\mathscr{E}(\mathcal{K})}.$$

Description of GAME_6**.** At this point, we replace by independent random values, all the values of k_1 computed by partnered honest terminals and the test card $\mathcal{C}^*$, in any session in which the KDF was queried on some honest input $(\mathsf{Z}_1, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}))$ appearing in any of the q_t test sessions. This can be done without loss of security: the fact that k_1 is essentially random to the adversary is guaranteed after GAME_2, while GAME_5 ensures that all the outputs for distinct values $(\mathsf{Z}_1, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}))$ in the test sessions are independent. As in GAME_4 we have no security loss.

Description of GAME_7**.** Finally in this game hop, the test card always encrypts certificate cert_0^*. We reduce a successful adversary distinguishing GAME_6 to GAME_7 to a multi-IND-CPA game, where each instance of the IND-CPA game is initialized with a value of k which is independent and chosen at random (this indeed fits our situation, since GAME_6 guarantees that all the computed k_1 values are independent). By a hybrid argument, we can show that for every adversary $\mathcal{A}$ playing a multi-IND-CPA game running in time $t_{\mathcal{A}}$ with q_t sessions there exists an adversary $\mathcal{B}$ playing the single-instance IND-CPA game running in time $t_{\mathcal{B}} = t_{\mathcal{A}} + O(q_t)$, such that it holds that: $\mathbf{Adv}^{\mathsf{multi\text{-}IND\text{-}CPA}}_{\mathsf{AES}}(t, q_t) \leq q_t \cdot \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AES}}(t + O(q_t))$. In particular, we can write:

$$\Pr[\text{GAME}_6] \leq \Pr[\text{GAME}_7] + q_t \cdot \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AES}}(t + O(q_t)).$$

At this point the adversary cannot distinguish between the use of cert_0^* (for $b = 0$) and cert_1^* (for $b = 1$) with better-than-guessing probability, because in both cases cert_0^* is encrypted now. Thus,

$$\Pr[\text{GAME}_7] \leq \frac{1}{2}.$$

This complete the proof of identity hiding. □

4.5 Deniability of O-FS

The theorem below shows that O-FS satisfies outsider deniability, but only in the restricted scenario which does not allow renegotiation.

Theorem 4.5 (Deniability of O-FS) *In the standard model, O-FS without persistent binding is outsider-deniable.*

In the following proof, we assume that parties' certificates are stored in a public list. This implicitly gives the simulator (and the adversary as well) access to static public keys. As the result holds for O-FS without persistent binding, we let each user ignore the control byte sent by its peer. Observe that the protocol is designed in a way that makes the terminal neglect its own requests (to use the PB mode) whenever the card refuses to do so. Having this scenario in mind, we can assume wlog. that a terminal always demands to renegotiate while the card always declines.

Proof. Intuitively, the reason why the protocol's transcripts are simulatable is that each message is either computable without knowledge of any long-term secret or indistinguishable from values randomly chosen from a suitable space.

Consider two honest users $\mathcal{T}$ and $\mathcal{C}$. We need to describe how the simulator computes messages indistinguishable from honest transcripts. In O-FS, the messages sent from the terminal

are (i) its certificate $\mathrm{cert}_{\mathcal{T}}$, (ii) an ephemeral public key $\mathsf{epk}_{\mathcal{T}}$ and (iii) its control byte $\mathtt{CB}_{\mathcal{T}}$. From the card side, we have (iv) otID, (v) $\mathsf{OpaqueData}$, (vi) $\mathsf{authcrypt}$ and (vii) the control byte of the card $\mathtt{CB}_{\mathcal{C}}$. In addition, (viii) the session keys must be simulatable. As discussed above, we assume that the terminal asks for renegotiation, but the card does not allow. This indeed eases the simulator's computation, as it can simply set the control bytes of terminal and card $\mathtt{CB}_{\mathcal{T}}$ and $\mathtt{CB}_{\mathcal{C}}$ accordingly.

In the simulator's description which follows we denote values that $\mathcal{S}$ computes by adding * in superscript. This allows to explicitly distinguish simulated values from "real" values.

Simulator $\mathcal{S}(\mathsf{ID}_{\mathcal{T}}, \mathsf{ID}_{\mathcal{C}})$:

1. Set $\mathtt{CB}^\star_{\mathcal{T}}$ and $\mathtt{CB}^\star_{\mathcal{C}}$;
2. Obtain certificates $\mathrm{cert}_{\mathcal{T}}$, $\mathrm{cert}_{\mathcal{C}}$;
3. Generate ephemeral keys for the terminal $(\mathsf{esk}^\star_{\mathcal{T}}, \mathsf{epk}^\star_{\mathcal{T}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$;
4. Generate ephemeral keys for the card $(\mathsf{esk}^\star_{\mathcal{C}}, \mathsf{epk}^\star_{\mathcal{C}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$;
5. Extract long-term public key $\mathsf{pk}_{\mathcal{T}}$ from $\mathrm{cert}_{\mathcal{T}}$.
6. Compute $\mathsf{Z}^\star_1 \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{esk}^\star_{\mathcal{C}}, \mathsf{pk}_{\mathcal{T}})$;
7. Compute intermediate keys $(k^\star_1, k^\star_2) \leftarrow \mathsf{KDF}\,(\mathsf{Z}^\star_1, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}^\star_{\mathcal{C}}))$;
8. Compute encrypted value $\mathsf{OpaqueData}^\star \leftarrow \mathsf{AES}_{k^\star_1}(\mathrm{cert}_{\mathcal{C}})$ and set $\mathsf{otID}^\star := \mathsf{epk}^\star_{\mathcal{C}}$;
9. Compute $\mathsf{Z}^\star \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{esk}^\star_{\mathcal{T}}, \mathsf{pk}_{\mathcal{C}})$;
10. Compute values $\mathsf{sk}^\star_{\mathsf{cfrm}}, \mathsf{sk}^\star_{\mathsf{MAC}}, \mathsf{sk}^\star_{\mathsf{Enc}}, \mathsf{sk}^\star_{\mathsf{RMAC}}$ by invoking $\mathsf{KDF}(\mathsf{Z}^\star, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{otID}^\star_{|_{1..8}}, \mathsf{epk}^\star_{\mathcal{T}}|_{1..16}, k^\star_2))$;
11. Compute tag $\mathsf{authcrypt}^\star \leftarrow \mathsf{CMAC}_{\mathsf{sk}^\star_{\mathsf{cfrm}}}(\mathtt{"KC\_1\_V"} \| \mathsf{otID}^\star_{|_{1..8}} \| \mathsf{ID}_{\mathcal{T}} \| \mathsf{epk}^\star_{\mathcal{T}}|_{1..16})$;
12. Return $(\mathrm{cert}_{\mathcal{T}}, \mathsf{epk}^\star_{\mathcal{T}}, \mathtt{CB}^\star_{\mathcal{T}})$ as the terminal's output, $(\mathsf{OpaqueData}^\star, \mathsf{authcrypt}^\star, \mathtt{CB}^\star_{\mathcal{C}}, \mathsf{otID}^\star)$ for the card's, and session keys $(\mathsf{sk}^\star_{\mathsf{MAC}}, \mathsf{sk}^\star_{\mathsf{Enc}}, \mathsf{sk}^\star_{\mathsf{RMAC}})$.

Observe that the simulator computes messages exactly as the terminal and the card would except for the shared secret $\mathsf{Z}^\star$. In the actual protocol, the card honestly obtains Z by using the terminal's ephemeral public key $\mathsf{pk}_{\mathcal{T}}$ and its own secret key $\mathsf{esk}_{\mathcal{C}}$. Since the simulator does not know the card's secret key (fixed in the KE experiment), but does know ephemeral keys of both card and terminal (it can choose these values itself), it can compute 'the right value' $\mathcal{DH}_{\mathscr{E}}(\mathsf{pk}_{\mathcal{C}}, \mathsf{esk}^\star_{\mathcal{T}}) = \mathcal{DH}_{\mathscr{E}}(\mathsf{sk}_{\mathcal{C}}, \mathsf{epk}^\star_{\mathcal{T}})$. The only difference between real and simulated executions is that, in the latter, the simulator picks fresh ephemeral keys: all the other values are derived 'honestly' from public values and these keys. Since ephemeral keys are freshly chosen at random for any new execution and not given to the adversary, she has no way to check whether they come from a real execution or from a simulation of it. □

The proof can be read by observing that an outsider, being restricted not to interfere with the parties involved in a KE execution, does not have enough information to convince a third party that the protocol execution between two given users $\mathcal{T}$ and $\mathcal{C}$ actually took place. She could indeed have produced fake (yet authentic-looking) transcripts by herself. Note, however, that the simplicity of the argument above is due to the weakness of the notion of deniability against outsiders.

Salvaging Untraceability. We have already shown how the use of persistent binding makes cards traceable. A natural question is whether neglecting the renegotiation mode could, as in the case of deniability, overcome the problem. Concerning O-FS without persistent binding, we observe that untraceability does hold and, moreover, it follows almost immediately from the fact that the protocol is outsider-deniable.

We stress again that, in general, the notion of untraceability is not implied by deniability. However, in the case of O-FS with no renegotiation, untraceability follows from the above

proof of outsider deniability. By looking at the protocol transcripts, an adversary cannot learn whether the transcript was simulated or generated honestly. Furthermore, two simulated runs of the protocol involving the same terminal $\mathcal{T}$ but two different cards $\mathcal{C}_0$ and $\mathcal{C}_1$ are indistinguishable. Indeed, the only message that could partially reveal information regarding which one of the two cards is running the protocol is the encryption of the card's certificate. On the other hand, as a ciphertext hides the content of its underlying plaintext, it should be hard to distinguish encryptions of $\text{cert}_{\mathcal{C}_0}$ and $\text{cert}_{\mathcal{C}_1}$, thus simulated transcripts involving a given terminal $\mathcal{T}$ and either $\mathcal{C}_0$ or $\mathcal{C}_1$ look almost the same. Hence, it follows from deniability that also the corresponding honest executions between $\mathcal{T}$ and $\mathcal{C}_0$, or $\mathcal{T}$ and $\mathcal{C}_1$, are indistinguishable: $\langle \mathcal{T}, \mathcal{C}_0 \rangle \approx \mathcal{S}(\mathcal{T}, \mathcal{C}_0) \approx \mathcal{S}(\mathcal{T}, \mathcal{C}_1) \approx \langle \mathcal{T}, \mathcal{C}_1 \rangle$, as untraceability requires.

5 Security of the Channel Protocol

Here we discuss briefly the security of the secure messaging (used both in ZKM and FS) and of the composition of the channel with the key agreement step.

Secure Messaging. Once the keys are generated the parties use them to secure the communication. The description [30] proposes two modes, one for command and response MACs without confidentiality (using keys $\mathsf{sk}_{\mathsf{MAC}}$ and $\mathsf{sk}_{\mathsf{RMAC}}$, respectively), and the other one for encrypted data transfer under the key $\mathsf{sk}_{\mathsf{Enc}}$ used by both parties. If only authenticity is required, then the data is secured according to ISO 7816-4 [31]; in case encryption is used the protocol basically follows the encrypt-then-MAC approach, first encrypting the payload.

Alarmingly, according to the standard [30], the terminal can ask the card via the control byte to only create a single key $\mathsf{sk}_{\mathsf{Enc}} = \mathsf{sk}_{\mathsf{RMAC}} = \mathsf{sk}_{\mathsf{MAC}}$, operating in a special mode (`ONE_SK`). Sharing the key among different primitives usually needs a cautionary treatment. It remains unclear why OPACITY implements this mode, but it does not seem to be recommendable from a pure security perspective. In what follows we assume that independent keys are used instead.

Encryption for the encrypt-then-MAC approach in the secure messaging is not further specified in [30]. The (outdated) implementation relies on AES encryption with a prepended, fixed-length session counter. The counter is implemented as a type `short` which usually limits the range to two bytes. The parties' counter values seem to be synchronized, yet the recipient seems to increment the counter before the MAC is actually verified and does not perform any check on the counter value himself (beyond the verification through the MAC). The message to be encrypted is first padded.

For authentication the parties first pad the message (or ciphertext) according to ISO 7816-4, basically prepending a MAC chaining value of 16 bytes before computing an AES-based CMAC [8, 34] according to SP 800-38B [49]. Only the first 8 bytes of the output are attached as the authentication code. Yet, the full 16 bytes are stored locally and used as the MAC chaining value for the next computation. Interestingly, while the approach is perfectly compliant with ISO 7816-4, the standard SP 800-38B actually suggests to use sequence numbers, time stamps, or nonces for protecting against replay attacks [49]. Potentially, the session counter used for encryption could have been used here. Also note that command MACs and response MACs are treated slightly differently, in addition to using different keys, but nonetheless follow the same pattern outlined here.

We omit a formal analysis of secure messaging which, except for the single-key mode, follows the common cryptographic approaches. It would be nonetheless interesting to provide such an analysis, taking into account recent attacks and models for such steps [37, 2, 55, 56]. However, it is beyond our scope here.

Composition. Clearly, a secure key-exchange protocol and secure messaging on their own may not be enough to ensure the security of the composed protocol. Several approaches exist to bridge this gap, ranging from monolithic analysis of the composed protocol, to general-purpose compositional frameworks like Canetti's Universal Composition (UC) model [15]. The latter has been successfully applied to analyze and construct securely composable key-exchange protocols [16]. However, security of key exchange in the UC model (and comparable simulation-based frameworks [40]) already imposes strong requirements on the protocols which are hard to meet.

Recent attempts by Brzuska et al. [14, 12] aim to provide compositional guarantees for game-based notions of key exchange, in particular, for BR-secure protocols. Roughly, they show that if the key exchange protocol is BR-secure and provides a property called public session matching, then the derived keys securely replaces fresh random keys in any symmetric-key protocol. The public session matching here allows one at any point to identify sessions which agree on the same key, given the public but adversarial controlled network communication of all concurrently executed protocol runs. Brzuska et al. [14] discuss a counter example where the adversary can re-randomize the encrypted transmissions internally such that matching becomes infeasible.

For O-FS we can deduce from our result about BR security and assuming secure messaging, that the composition of the two steps is also secure *if public session matching holds*. This property is given here because one can match outgoing messages of the terminal with ingoing messages to the card directly (i.e., $\mathsf{cert}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}$ are sent in clear), and vice versa the card's message contains the "unique identifier" $\mathsf{authcrypt}$ which the adversary cannot change (e.g., re-randomize) when delivering to the terminal, without making the terminal reject. Hence, public session matching holds and, assuming secure messaging, therefore security of the composed protocol in the sense of [14] also holds.

For O-ZKM the protocol cannot be shown to be BR-secure, because the derived encryption key $\mathsf{sk}_{\mathsf{Enc}}$ is already used in the key exchange step to encrypt the GUID. Hence, the composition result in [14] does not apply. In principle, using a concurrent work of Brzuska et al. [12] one could argue about the security of the composed protocol in such cases. Since we do not recommend the use O-ZKM in the first place we omit such an analysis.

6 Conclusion

Our analysis reveals that, from a cryptographic point of view, O-FS achieves a decent level of key secrecy, but has clear restrictions on privacy guarantees. For one, privacy could be improved by also encrypting the card's control byte $\mathtt{CB}_{\mathcal{C}}$ for persistent binding, hiding the fact if the card has been used in connection with that terminal before. Whereas the situation for O-FS is arguable, we do not recommend O-ZKM for deployment. This is due to its rather weak security guarantees for (terminal) authentication and the weaker form of identity hiding.

Our analysis also shows common problems in making precise security claims about real protocols. Like with every cryptographic (or scientific) model we have to abstract out some details. This can be an impediment in particular in view of the fact that the protocol can operate in various modes, e.g., for compatibility reasons. This complexity is the cryptographer's enemy, discussing all possibilities is often beyond a reasonable approach. However, omitting some of these modes is dangerous, as they often admit back doors for attackers. There are some *potential* back doors for OPACITY as well, e.g., the single-key mode $\mathtt{ONE\_SK}$ for secure messaging. This is magnified by the fact that OPACITY is not fully specified with respect to all relevant details (e.g., which encryption mode is used for $\mathsf{OpaqueData}$). Also, the binding of sessions to their keys is rather loose as merely a partial transcript of the execution enters the key derivation resp. message authentication. In this sense, it should be understood that our (partly positive) cryptographic analysis has its inherent limitations.

Acknowledgments

Marc Fischlin was supported by a Heisenberg grant Fi 940/3-1 of the German Research Foundation (DFG). This work was supported by CASED (`www.cased.de`) and by EC SPRIDE.

References

[1] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 65–84, Les Diablerets, Switzerland, January 23–26, 2005. Springer, Berlin, Germany. (Cited on page 7.)

[2] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26, Oakland, California, USA, May 17–20, 2009. IEEE Computer Society Press. (Cited on page 28.)

[3] Smart Card Alliance. Industry technical contributions: Opacity. `http://www.smartcardalliance.org/pages/smart-cards-contributions-opacity`, April 2013. (Cited on pages 1 and 2.)

[4] ANS. ANSI x9.63-199x – public key cryptography for the financial services industry: Key agreement and key transport using elliptic curve cryptography, 1999. (Cited on pages 3 and 15.)

[5] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Berlin, Germany. (Cited on pages 3 and 5.)

[6] Jens Bender, Özgür Dagdelen, Marc Fischlin, and Dennis Kügler. Domain-specific pseudonymous signatures for the german identity card. In Dieter Gollmann and Felix C. Freiling, editors, *ISC 2012: 15th International Conference on Information Security*, volume 7483 of *Lecture Notes in Computer Science*, pages 104–119, Passau, Germany, September 19–21, 2012. Springer, Berlin, Germany. (Cited on page 4.)

[7] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009: 12th International Conference on Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 33–48, Pisa, Italy, September 7–9, 2009. Springer, Berlin, Germany. (Cited on page 37.)

[8] John Black and Phillip Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. *Journal of Cryptology*, 18(2):111–131, April 2005. (Cited on page 28.)

[9] Colin Boyd, Wenbo Mao, and Kenneth G. Paterson. Deniable authenticated key establishment for internet protocols. In *Security Protocols Workshop*, pages 255–271, 2003. (Cited on page 4.)

[10] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*. The MIT Press, 2000. (Cited on page 4.)

[11] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 04: 11th Conference on Computer and Communications Security*, pages 132–145, Washington D.C., USA, October 25–29, 2004. ACM Press. (Cited on page 4.)

[12] C. Brzuska, M. Fischlin, N.P. Smart, B. Warinschi, and S. Williams. Less is more: Relaxed yet composable security notions for key exchange. Cryptology ePrint Archive, Report 2012/242, 2012. `http://eprint.iacr.org/`. (Cited on pages 4, 29, and 37.)

[13] Christina Brzuska, Özgür Dagdelen, and Marc Fischlin. Tls, pace, and eac: A cryptographic view at modern key exchange protocols. In *Sicherheit*, pages 71–82, 2012. (Cited on page 37.)

[14] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 51–62, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. (Cited on pages 4 and 29.)

[15] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press. (Cited on page 29.)

[16] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer, Berlin, Germany. (Cited on pages 3, 8, and 29.)

[17] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM*, 28, October 1985. (Cited on page 4.)

[18] Cas Cremers and Michele Feltz. One-round strongly secure key exchange with perfect forward secrecy and deniability. Cryptology ePrint Archive, Report 2011/300, 2011. `http://eprint.iacr.org/`. (Cited on pages 10 and 11.)

[19] Özgür Dagdelen and Marc Fischlin. Security analysis of the extended access control protocol for machine readable travel documents. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC 2010: 13th International Conference on Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 54–68, Boca Raton, FL, USA, October 25–28, 2010. Springer, Berlin, Germany. (Cited on page 37.)

[20] Matthias Deeg, Christian Eichelmann, and Sebastian Schreiber. Programmed insecurity - syss cracks yet another usb flash drive. `http://www.syss.de/fileadmin/ressources/040_veroeffentlichungen/dokumente/SySS_Cracks_Yet_Another_USB_Flash_Drive.pdf`. (Cited on page 2.)

[21] Matthias Deeg and Sebastian Schreiber. Cryptographically secure? syss cracks a usb flash drive. `https://www.syss.de/fileadmin/ressources/040_veroeffentlichungen/dokumente/SySS_Cracks_SanDisk_USB_Flash_Drive.pdf`. (Cited on page 2.)

[22] Mario Di Raimondo and Rosario Gennaro. New approaches for deniable authentication. *Journal of Cryptology*, 22(4):572–615, October 2009. (Cited on pages 4, 10, and 11.)

[23] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Deniable authentication and key exchange. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 400–409, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. (Cited on pages 4, 10, and 11.)

[24] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 146–162. Springer, Berlin, Germany, March 15–17, 2009. (Cited on pages 4, 10, and 11.)

[25] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004. (Cited on pages 4 and 10.)

[26] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *ProvSec 2008: 2nd International Conference on Provable Security*, volume 5324 of *Lecture Notes in Computer Science*, pages 313–327, Shanghai, China, October 31 – November 1, 2008. Springer, Berlin, Germany. (Cited on page 2.)

[27] Ian Goldberg. On the security of the tor authentication protocol. In *Privacy Enhancing Technologies*, pages 316–331, 2006. (Cited on page 36.)

[28] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. `http://www.cacr.math.uwaterloo.ca/techreports/2011/cacr2011-11.pdf`, 2012. Online first; print version to appear. (Cited on pages 4 and 36.)

[29] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. (Cited on page 10.)

[30] INCITS. 504-1, information technology - generic identity command set part 1: Card application command set. (Cited on pages 1 and 28.)

[31] ISO/IEC. Identification cards - Integrated circuit(s) cards with contacts - Part 4: Organization, security and commands for interchange. Technical Report ISO/IEC 7816-4, International Organization for Standardization, Geneva, Switzerland, 2005. (Cited on page 28.)

[32] ISO/IEC. Identification cards - Integrated circuit(s) cards with contacts - Part 8: Security related interindustry commands. Technical Report ISO/IEC 7816-8, International Organization for Standardization, Geneva, Switzerland, 2009. (Cited on page 13.)

[33] ISO/IEC. Identification Cards Integrated Circuit Cards Programming Interface Part 6: Registration procedures for the authentication protocols for interoperability. Technical Report ISO/IEC FDIS 24727-6, International Organization for Standardization, Geneva, Switzerland, 2009. (Cited on page 1.)

[34] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153, Lund, Sweden, February 24–26, 2003. Springer, Berlin, Germany. (Cited on page 28.)

[35] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in*

Cryptology – CRYPTO 2012, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Germany. (Cited on page 2.)

[36] Ari Juels and Stephen A. Weis. Defining strong privacy for RFID. Cryptology ePrint Archive, Report 2006/137, 2006. `http://eprint.iacr.org/`. (Cited on pages 3 and 9.)

[37] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany. (Cited on page 28.)

[38] Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Germany. (Cited on page 15.)

[39] Caroline Kudla and Kenneth G. Paterson. Modular security proofs for key agreement protocols. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 549–565, Chennai, India, December 4–8, 2005. Springer, Berlin, Germany. (Cited on pages 3 and 15.)

[40] Ralf Küsters and Max Tuengerthal. Composition theorems without pre-established session identifiers. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 41–50, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. (Cited on page 29.)

[41] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16, Wollongong, Australia, November 1–2, 2007. Springer, Berlin, Germany. (Cited on pages 3, 8, and 36.)

[42] Kristin Lauter and Anton Mityagin. Security analysis of KEA authenticated key exchange protocol. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 378–394, New York, NY, USA, April 24–26, 2006. Springer, Berlin, Germany. (Cited on pages 3 and 15.)

[43] Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas, and Scott A. Vanstone. An efficient protocol for authenticated key agreement. *Des. Codes Cryptography*, 28(2):119–134, 2003. (Cited on page 15.)

[44] Alfred Menezes and Berkant Ustaoglu. Security arguments for the UM key agreement protocol in the NIST SP 800-56A standard. In Masayuki Abe and Virgil Gligor, editors, *ASIACCS 08: 3rd Conference on Computer and Communications Security*, pages 261–270, Tokyo, Japan, March 18–20, 2008. ACM Press. (Cited on pages 3 and 15.)

[45] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 55–73, Melbourne, Australia, December 7–11, 2008. Springer, Berlin, Germany. (Cited on page 36.)

[46] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. The TLS handshake protocol: A modular analysis. *Journal of Cryptology*, 23(2):187–223, April 2010. (Cited on page 2.)

[47] NIST. SKIPJACK and KEA Algorithm Specifications. Technical report, National Institute of Standards and Technology, 1998. (Cited on page 15.)

[48] NIST. Security Requirements for Cryptographic Modules. Technical Report FIPS 140-2, National Institute of Standards and Technology, 2002. (Cited on page 2.)

[49] NIST. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. Technical Report SP 800-38B, National Institute of Standards and Technology, 2007. (Cited on pages 14 and 28.)

[50] NIST. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. Technical Report SP800-56A, National Institute of Standards and Technology, 2007. (Cited on pages 2 and 14.)

[51] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118, Cheju Island, South Korea, February 13–15, 2001. Springer, Berlin, Germany. (Cited on pages 4 and 17.)

[52] OPACITY. Reference Implementation - `sourceforge.net/projects/opacity/`. `sourceforge.net/projects/opacity/`. (Cited on page 2.)

[53] Khaled Ouafi and Raphael C.-W. Phan. Privacy of recent rfid authentication protocols. In *ISPEC*, pages 263–277, 2008. (Cited on pages 3 and 9.)

[54] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 316–337, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Berlin, Germany. (Cited on page 10.)

[55] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389, Seoul, South Korea, December 4–8, 2011. Springer, Berlin, Germany. (Cited on pages 2 and 28.)

[56] Kenneth G. Paterson and Gaven J. Watson. Authenticated-encryption with padding: A formal security treatment. In *Cryptography and Security*, volume 6805 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2012. (Cited on page 28.)

[57] Eric F. Le Saint and Dominique Louis Joseph Fedronic. Open protocol for authentication and key establishment with privacy, 07 2010. (Cited on page 1.)

[58] Eric Le Saint. Opacity - the new open protocol of choice. `http://www.itsecurityhub.eu/2012/08/opacity-the-new-open-protocol-of-choice/`, August 2012. (Cited on pages 1 and 2.)

[59] Eric Le Saint. Personal communication, July 2012. (Cited on page 2.)

[60] Eric Le Saint, Dom Fedronic, and Steven Liu. Open protocol for access control identification and ticketing with privacy. `http://www.smartcardalliance.org/resources/pdf/OPACITY_Protocol_3.7.pdf`, July 2011. (Cited on pages 3 and 12.)

[61] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Berlin, Germany. (Cited on page 2.)

[62] Serge Vaudenay. On privacy models for RFID. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 68–87, Kuching, Malaysia, December 2–6, 2007. Springer, Berlin, Germany. (Cited on page 9.)

[63] Andrew C. Yao and Yunlei Zhao. A new family of implicitly authenticated diffie-hellman protocols. Cryptology ePrint Archive, Report 2011/035, 2011. `http://eprint.iacr.org/`. (Cited on pages 10 and 15.)

[64] Andrew Chi-Chih Yao and Yunlei Zhao. Deniable internet key exchange. In Jianying Zhou and Moti Yung, editors, *ACNS 10: 8th International Conference on Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 329–348, Beijing, China, June 22–25, 2010. Springer, Berlin, Germany. (Cited on page 10.)

A Security Analysis of O-ZKM

A.1 Ambiguities in the O-ZKM Standard

Before discussing the security guarantees provided by O-ZKM we need to remark on two problems due to ambiguous descriptions, or misspecification, in the standard. The first problem occurs in line 45 (see the protocol description on page 42) where the terminal supposedly should recover the card's identity GUID from the certificate. However, in case persistent binding is not used, the card always sends the blinded certificate (i.e., with GUID removed) and hence GUID cannot be recovered given the blinded certificate $\mathsf{cert}^*_{\mathcal{C}}$ only. If, on the other hand, persistent binding is used, then the card only sends the first 8 bytes of the hash value of its blinded certificate, i.e., $\mathsf{HASH}(\mathsf{cert}^*_{\mathcal{C}})_{|1..8}$ and, thus, again there is no way for the terminal to recover the card's GUID. As value GUID is, surprisingly, never used by the terminal anyways, we simply treat line 45 as non-existent.

The second, more dangerous problem is in line 46 where the terminal supposedly validates the card's certificate. The issue is that, at this point, the terminal may *not* have computed the certificate. In case persistent binding is used, the card only transmits the truncated hash of its blinded certificate and, thus, the terminal can in no way recover the corresponding certificate. On the other hand, in case persistent binding is not used, only the blinded certificate is sent to the terminal: to recover the certificate, the terminal is dependent on the card sending its GUID (encrypted) in addition to the blinded certificate. A malicious card may, however, not always do this. A cheap fix to this problem might be to not check certificates in the persistent binding branch and to treat the incapability of the terminal to recover the certificate as evidence that the certificate is invalid, in case persistent binding is not used. Yet, the problem is not as readily fixable as will become apparent given the following attack.

Consider the protocol with the two fixes as described above. Then a malicious card can mount the following attack to authenticate towards an honest terminal. First, it chooses a key pair $\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{C}}$ and generates an (invalid) certificate containing its public key $\mathsf{pk}_{\mathcal{C}}$. It then engages

a terminal in a first authentication session, which will eventually reject as the certificate is invalid. However, during the process, the terminal's computation includes a value nextZ which can also be computed by the malicious card. Furthermore, the terminal stores the value in its persistent binding registry taking the hash of the fake certificate as look-up key. Thus, in a second session with the same terminal, the card can now authenticate using persistent binding as its certificate won't be checked anymore.

To circumvent the just described attack we assume the following fix. In case of persistent binding, we assume that no certificate is validated. Otherwise, if persistent binding is not used and the terminal cannot validate the certificate (due either to its invalidity or to the terminal not being able to recover the certificate) we assume that the terminal removes the previously added entry from its persistent binding registry. Note also that this prevents a trivial "flooding" attack where a malicious card injects invalid entries into the terminal's registry.

In any case, the ambiguities with respect to the description of O-ZKM are alarming, as this may lead to flawed implementations. The following (already weak) security guarantees should thus only be taken with another grain of salt.

A.2 Key Secrecy

In this section, we analyze the security of O-ZKM as an authenticated key agreement. Unfortunately, we cannot show security with respect to the BR model as described in Section 2. This is due to the fact that here only the chip card authenticates during a protocol execution: as terminals are not required to authenticate, they have no assigned secrets. These kind of protocols are also called *one-way authenticated key agreement protocols*. To our knowledge, there is only one security model considering one-way authentication due to Goldberg et al. [28], while few others are tailor-made for the respective analyzed protocols (e.g., [27] for Tor, and [45] for TLS). We observe that O-ZKM cannot be proven secure in the model of Goldberg et al. [28], since it is based on the eCK model [41] which, unlike BR, considers leakage of ephemeral keys. Neither O-FS nor O-ZKM is resistant against such leakage.

Therefore, our analysis of O-ZKM relies on a different approach. We adhere to the BR model, but slightly modify the notion of freshness of a session by requiring one more condition. That is, we call a session *ZKM-fresh* if the conditions of the BR model are satisfied and if the session is derived by an Execute-query. Basically, this means that we only consider passive adversaries. This additional restriction guarantees that neither the terminal nor the card are adversarially controlled and that any random choice is made by honest parties. This is inevitable in order to avoid the following attack: the adversary sends her message to an arbitrary card on behalf of an honest terminal (note that terminals can be considered honest because they do not authenticate and, thus, the adversary does not need to corrupt any terminal in order to successfully impersonate it); hence, she computes the session keys in this session and then queries the partner session to the Test-oracle. As neither Reveal nor Corrupt-queries took place, the partner session is fresh. Moreover, terminal and chip card are not controlled by the adversary. Finally, as the adversary computed the same session keys as in the tested session, she trivially wins the AKE experiment.

In O-ZKM we encounter another obstacle. Security proofs in the BR model provide strong security properties; in particular, session keys derived by BR-secure KE protocols are indistinguishable from suitable values uniformly sampled from the key domain. However, this cannot be ensured if (part of) the session key is already used in the execution, for instance, to confirm the knowledge of secrets via message authenticated codes. An adversary would simply pick the session key given by the Test-oracle and check its validity by re-performing the computation of a MAC as in the actual execution. Thereby, the adversary trivially tells apart genuine session

keys from random ones.

Key confirmation takes place in various KE protocols and is handled in different ways. For instance, the key exchange protocols PACE and EAC deployed in the current German electronic ID card were analyzed for a modified version such that the key derivation function outputs another distinct key only for the purpose of key confirmation [7, 19]. In the random-oracle model, the knowledge of this additional ephemeral key does not help the adversary to win the experiment. If the messages to be processed within the KE step differ in structure with the ones processed in the subsequent secure channel protocol, then the actual protocol is secure if the modified one is.

An alternative approach is proposed by Brzuska et al. [12] where the authors show that a relaxed notion of BR provides security if one further shows that the derived keys are suitable for the cryptographic primitives used in the following channel protocol. That is, session keys can be used already in the key exchange step, as long as this does not interfere with the subsequent deployment; in our case if the channel protocol ensures that the values encrypted have a different format than the one encrypted in the key exchange step. For more details on both approaches, we refer to [13].

For our analysis, we use the former approach and let the key derivation function (KDF) output an additional key $\mathsf{sk}_{\mathsf{EncG}}$ which is used to encrypt the GUID of the card. Thus, we replace line 14 and 37 in the ZKM protocol (cf. Figure 3) by

$$(\mathsf{sk}_{\mathsf{cfrm}}, \mathsf{sk}_{\mathsf{MAC}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{EncG}}, \mathsf{sk}_{\mathsf{RMAC}}, \mathsf{nextZ}) \leftarrow \mathsf{KDF}(\mathsf{Z}, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{C}}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}|_{1..16}}, \mathrm{n}_{\mathcal{C}})),$$

and line 21 (resp. line 44) by

$$\mathsf{encGUID} := \mathsf{GUID} \oplus \mathsf{AES}_{\mathsf{sk}_{\mathsf{EncG}}}(IV) \quad (\text{resp. } \mathsf{GUID} := \mathsf{encGUID} \oplus \mathsf{AES}_{\mathsf{sk}_{\mathsf{EncG}}}(IV)).$$

The following theorem refers to such a modified protocol, that we denote Π'_{OZKM}. We implicitly assume that the key derivation function (KDF) and any hash function used within the protocol are modeled as random oracles.

For the key-secrecy proof we consider sessions as indicated in Section 2, such that the session id sid for Π'_{OZKM} is set as $\mathsf{sid} = (\mathsf{HASH}(\mathrm{cert}^*_{\mathcal{C}})_{|_{1..8}}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}|_{1..16}, \mathrm{n}_{\mathcal{C}})$; the partner id pid is set to empty on the card's side since the terminal does not authenticate to the chip. Vice versa, the partner id pid is set to GUID on the terminal's side. Note that including the nonces in the session identifiers essentially makes them for executions with honest cards unique. In this sense, there is most likely at most one partnered session with the same KDF output (unless the KDF accidentally creates collisions).

Theorem A.1 (Key Secrecy of OPACITY Zero-Key Management) *In the random oracle model, we have*

$$\mathbf{Adv}^{ake}_{\Pi'_{\mathsf{OZKM}},\mathcal{A}} \leq 2q_e^2 \cdot \mathbf{Adv}^{\mathrm{GDH}}_{\mathcal{A}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h) + \frac{q_e(2q_e + q_h)}{2^{\ell_{\mathsf{Z}}}}$$

where λ denotes the security parameter, t the running time of adversary $\mathcal{A}$, q_e (resp. q_h) the number of executions (resp. hash computations), and ℓ_{Z} denotes the bit length of the Z.

We note that both the protocols Π'_{OZKM} and Π_{OZKM} are, strictly speaking, not complete. That is, while the card uses the globally unique $\mathsf{ID}_{\mathcal{T}}$ to identify renegotiation data, and thus cannot start a wrong persistent binding session, the terminal recovers sessions via the 8 first bytes $\mathsf{ID}_{\mathcal{C}}$ of the hashed certificate. Assuming a decent hash function one may assume that these values are quasi unique, but there is nonetheless a probability of about $q_e^2 \cdot 2^{-64}$ that different cards yield the same identifier $\mathsf{ID}_{\mathcal{C}}$ of length $\ell_{\mathsf{ID}_{\mathcal{C}}} = 64$.

Proof. The following proof is similar to the proof of key secrecy for O-FS (see Theorem 4.2). We begin with GAME_0, the original attack against key secrecy.

Description of GAME_0**.** This corresponds to the original attack on the protocol Π'_{OZKM}.

Description of GAME_1**.** This game works as GAME_0 but now we abort the game if the adversary $\mathcal{A}$ makes a hash query to the KDF for a Diffie–Hellman key Z computed by an honest card in a *ZKM-fresh* session without persistent binding.

The reduction to the GDH problem follows as in the case of GAME_2 for O-FS (see Theorem 4.2):

$$\Pr[\text{GAME}_0] \leq \Pr[\text{GAME}_1] + q_e^2 \cdot \mathbf{Adv}^{\text{GDH}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h).$$

Note that, while in GAME_2 for O-FS adversary $\mathcal{A}_{\text{GDH}}$ injects the challenge in $\mathsf{epk}_{\mathcal{C}}$ and $\mathsf{pk}_{\mathcal{T}}$, we now inject the challenge in $\mathsf{epk}_{\mathcal{T}}$ and $\mathsf{pk}_{\mathcal{C}}$. The remainder follows analogously.

Description of GAME_2**.** This game works as GAME_1 but now we abort the game if the adversary $\mathcal{A}$ makes a hash query to the KDF for a Diffie–Hellman key Z computed by an honest terminal in a *ZKM-fresh* session without persistent binding.

The reduction to the GDH problem follows as in the case of GAME_1:

$$\Pr[\text{GAME}_1] \leq \Pr[\text{GAME}_3] + q_e^2 \cdot \mathbf{Adv}^{\text{GDH}}(t + O(\lambda \cdot q_e \log q_e), 2q_e + q_h).$$

The next game also covers the case of a renegotiation step:

Description of GAME_3**.** In this game we also consider the persistent binding branch. That is, this game works as GAME_2 but now we also abort the game, if the adversary $\mathcal{A}$ makes a hash query to the KDF for a Diffie–Hellman key Z computed by an honest card or an honest terminal in the single Test-session.

According to the previous games, the adversary never queries the KDF (in a ZKM-fresh session without persistent binding) about the input Z as computed by an honest terminal or card. Hence, from $\mathcal{A}$'s point of view the part nextZ in a fresh session, and thereby in the Test-session, is basically an unknown value. As the adversary makes at most q_h queries to the KDF (where we also have to take the at most $2q_e$ other distinct queries by honest parties into account), and there appear at most q_e many distinct values for nextZ the probability of finding a pre-image to nextZ can be bound by $2q_e(q_e + q_h) \cdot 2^{-\ell_{\mathsf{Z}}}$.

Thus, we have

$$\Pr[\text{GAME}_2] \leq \Pr[\text{GAME}_3] + \frac{q_e(2q_e + q_h)}{2^{\ell_{\mathsf{Z}}}}.$$

We have reached the point where no adversary queries the secret Z derived in a session of an honest party with a partner identity of an honest party. However, this still does not necessarily yield a secure key exchange protocol, since instead of computing the session keys, an adversary could enforce the same input to the KDF function with identical Z in another execution with an honest party. Then, if another honest session yielded the same input, a Reveal-query for this session would clearly help the adversary to distinguish the tested session key from random. We next argue why this event cannot happen. To this end recall that such a Reveal-query must be for an unpartnered session, i.e., with a different session id sid.

Description of GAME_4**.** This game works as GAME_3, but aborts if there are two honest parties with both accepting sessions yielding identical input to the KDF but which are not partnered.

Note that $\mathsf{sid} = (\mathsf{HASH}(\mathrm{cert}^*_{\mathcal{C}})_{|_{1..8}}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}|_{1..16}, \mathrm{n}_{\mathcal{C}})$ and that the function info is injective. Hence, the data from session identifiers enters the KDF evaluation (for both new and renegotiation steps) such that distinct sid's for unpartnered sessions yield distinct input to the KDF. Hence, any call to KDF by another honest unpartnered party is for a different input than in the test session. Put differently, the derived key in a tested session is independent from keys derived by other (unpartnered) parties.

$$\Pr[\mathrm{GAME}_3] \leq \Pr[\mathrm{GAME}_4].$$

Thus, we have eventually reached a game where the adversary can merely guess the secret bit b because no unpartnered sessions have the same KDF input and the adversary never queries the KDF function on the same input as an honest party does in a fresh session. Hence, $\Pr[\mathrm{GAME}_4] \leq \frac{1}{2}$. Summing up all probabilities for game transitions yields the claim for the original attack in GAME_0. □

(Weak) Forward Secrecy. It is easy to see that O-ZKM does not offer forward secrecy and not even weak forward secrecy. Indeed, when corrupting a card's instance and learning its long-term secret enables to compute the intermediate secret Z. Together with the nonce $\mathrm{n}_{\mathcal{C}}$ which is included in the transcript, an adversary can clearly compute the session keys and distinguish genuine session keys from random. Hence, O-ZKM is not (weakly) forward-secure.

On Further Desirable Security Properties. We cannot prove resistance against Key-Compromise Impersonation (KCI) attacks because the terminal does not authenticate at all. Since the terminal has no long-term secrets a malicious chip cannot take advantage of it to authenticate and/or compute the session key. Hence, O-ZKM does not provide security against KCI.

According to leakage of ephemeral secrets, we identify $\mathsf{esk}_{\mathcal{T}}$ as the only ephemeral secret in O-ZKM since the nonce $\mathrm{n}_{\mathcal{C}}$ is sent to the channel and, thus, is public. The knowledge of $\mathsf{esk}_{\mathcal{T}}$ suffices, however, to compute the intermediate secret Z and subsequently, compute the session key. Therefore, leakage of ephemeral break the protocol security as a key agreement totally.

A.3 Impersonation Resistance

Impersonation resistance of the original O-ZKM protocol (or our modified one for key secrecy) follows almost as in the case of O-FS. We have to take into account, though, that we have shown key secrecy for passively observed executions only, whereas impersonation resistance should hold for malicious cards. But note that, independently of whether the transmitted card's certificate in an impersonation attempt with an honest terminal contains GUID in encrypted form or as plain text, there can be at most one honest card carrying a valid certificate with this GUID; any other GUID for an unregistered or an adversarially-controlled card would otherwise violate the unforgeability of certificates. Since the terminal eventually checks the validity of the certificate this implies that for a successful impersonation the adversary needs to use the valid certificate of an honest card.[3] We next argue that the fact that the public key is certified basically requires to break the Gap Diffie-Hellman problem, or to forge a MAC.

If persistent binding is not used in an impersonation attempt, then the card certificate includes the genuine public key of the honest card and the argument that the adversary can never query the KDF about the value Z applies as in the proof of key secrecy under the Gap

[3]Recall that we assume that changes in the registry are undone again (or that the corresponding entry is deleted) if the eventual certificate verification fails, such that only entries for the genuine card are listed in further renegotiation steps (see Appendix A.1).

Diffie-Hellman assumption. If persistent binding is used, on the other hand, the adversary can only win if it manages to "confuse" the honest terminal in trying to reconnect with a different $\mathsf{ID}_{\mathcal{C}}$, possibly belonging to a session of a malicious card, or if it can compute nextZ for the correct value $\mathsf{ID}_{\mathcal{C}}$ of a previous run of the honest terminal and the honest card. Assuming that HASH behaves like a random oracle, the former happens with probability $q_e^2 \cdot 2^{-\ell_{\mathsf{ID}_{\mathcal{C}}}}$ only (for $\ell_{\mathsf{ID}_{\mathcal{C}}} \geq 64$); the latter would contradict the key secrecy (because entries in the registry for such a "good" $\mathsf{ID}_{\mathcal{C}}$ could have been only created with a run involving the genuine public key of the card, by our above assumption about registry updates, and could not have been computed by the adversary). Given that the adversary cannot derive the session key in an impersonation attempt, and that there is no unpartnered session with the same input to the KDF, the adversary needs to forge a MAC for an unknown key (as in the proof for O-FS). This shows impersonation resistance.

A.4 Privacy

Untraceability. The OPACITY protocol with Full Secrecy O-FS does *not* preserve untraceability. In fact, in Section 4.4 we showed two attacks that allow an adversary to tell two cards apart by using persistent binding. Both attacks are still valid for the OPACITY protocol with Zero-Key Management. The O-ZKM protocol is susceptible to another attack. In particular, cards can *always* be recognized and traced from terminal to terminal, since they send, as part of their response, the blinded identifier $\mathsf{blindID}$. If persistent binding is used, $\mathsf{blindID}$ is a constant value, i.e., the first eight bytes of the hash of the certificate; else, if persistent binding is *not* used, then the value is again constant consisting of the blinded certificate, i.e., the certificate from which the identity GUID was removed. Since there is no freshness introduced into the hash, a card $\mathcal{C}$ always sends in clear, to any terminal it visits, the same string $\mathsf{blindID}$. Thus, cards using the O-ZKM protocol are *never* untraceable. This attack is much stronger than the attacks against the O-FS protocol, enabling easy linkability between sessions of the same card.

Identity-Hiding. The protocol also does *not* achieve identity-hiding. Consider the following attack. We assume that the adversary knows the list of all certificates and thus the list of all valid GUIDs. On observing an honest execution the certificate is send in the clear except for the part that contains the GUID which is sent only encrypted. At this point it can simply try all possible GUID, insert each into the partial certificate and check if it validates. Thus in time $\mathcal{O}(\#\mathsf{GUID})$ (where $\#\mathsf{GUID}$ denotes the number of tested identities) an adversary can recover the identity of a card that was used in an honest execution.

A.5 Deniability of O-ZKM

With an argument analogous to the one we use to prove deniability of O-FS, we show that also O-ZKM satisfies outsider deniability. We stress that both results hold only conditionally, since we need to assume that parties never renegotiate. Do note, that unlike for the O-FS case untraceability does not follow for O-ZKM. That is, O-ZKM without persistent binding is outsider deniable, but *not* untraceable.

Theorem A.2 *In the standard model, O-ZKM without persistent binding is outsider-deniable.*

The idea behind the proof is as in Theorem 4.5, namely it is easy to simulate the transcripts of honest KE runs. Indeed, the simulator can compute messages indistinguishable from real transcripts by using public values (i.e., static public keys) and by computing the missing information by itself (after generating the single ephemeral key pair).

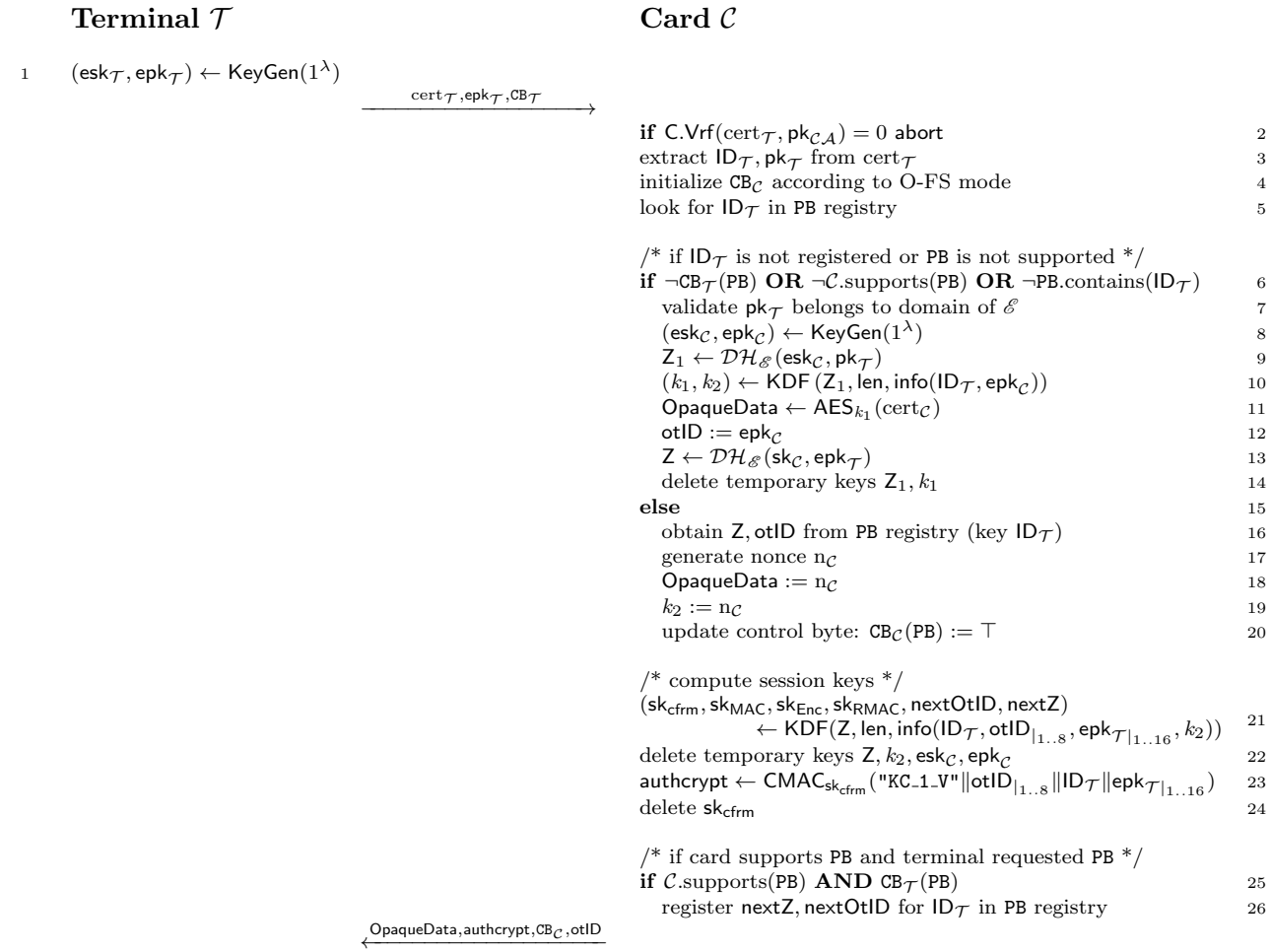

Figure 2: OPACITY with Full Secrecy

Terminal $\mathcal{T}$		Card $\mathcal{C}$	
1	$(\mathsf{esk}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$		
	$\xrightarrow{\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}, \mathtt{CB}_{\mathcal{T}}}$		
		$\mathsf{ID}_{\mathcal{C}} := \mathsf{HASH}(\mathrm{cert}^*_{\mathcal{C}})_{\vert 1..8}$	2
		initialize $\mathtt{CB}_{\mathcal{C}}$ according to O-ZKM mode	3
		look for $\mathsf{ID}_{\mathcal{T}}$ in PB registry	4
		/* if $\mathsf{ID}_{\mathcal{T}}$ is not registered or PB is not supported */	
		if $\neg\mathtt{CB}_{\mathcal{T}}(\mathtt{PB})$ **OR** $\neg\mathcal{C}$.supports(PB) **OR** $\neg$PB.contains($\mathsf{ID}_{\mathcal{T}}$)	5
		validate $\mathsf{epk}_{\mathcal{T}}$ belongs to domain of $\mathscr{E}$	6
		$\mathsf{Z} \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{sk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{T}})$	7
		$\mathsf{blindID} := \mathrm{cert}^*_{\mathcal{C}}$ /* see Section 3.1 */	8
		else	9
		obtain Z from PB registry (key $\mathsf{ID}_{\mathcal{T}}$)	10
		$\mathsf{blindID} := \mathsf{ID}_{\mathcal{C}}$	11
		update control byte: $\mathtt{CB}_{\mathcal{C}}(\mathtt{PB}) := \top$	12
		/* compute session keys */	
		generate nonce $\mathrm{n}_{\mathcal{C}}$	13
		$(\mathsf{sk}_{\mathsf{cfrm}}, \mathsf{sk}_{\mathsf{MAC}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{RMAC}}, \mathsf{nextZ})$ $\leftarrow \mathsf{KDF}(\mathsf{Z}, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{C}}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}\vert 1..16}, \mathrm{n}_{\mathcal{C}}))$	14
		delete Z	15
		$\mathsf{authcrypt} \leftarrow \mathsf{CMAC}_{\mathsf{sk}_{\mathsf{cfrm}}}(\mathtt{"KC\_1\_V"} \Vert \mathsf{ID}_{\mathcal{C}} \Vert \mathsf{ID}_{\mathcal{T}} \Vert \mathsf{epk}_{\mathcal{T}\vert 1..16})$	16
		delete $\mathsf{sk}_{\mathsf{cfrm}}$	17
		/* if card supports PB and terminal requested PB */	
		if $\mathcal{C}$.supports(PB) **AND** $\mathtt{CB}_{\mathcal{T}}(\mathtt{PB})$	18
		register nextZ for $\mathsf{ID}_{\mathcal{T}}$ in PB registry	19
		/* if reader requested GUID */	
		if $\mathtt{CB}_{\mathcal{T}}(\mathtt{RET\_GUID})$	20
		$\mathsf{encGUID} := \mathsf{GUID} \oplus \mathsf{AES}_{\mathsf{sk}_{\mathsf{Enc}}}(IV)$	21
		update control byte: $\mathtt{CB}_{\mathcal{C}}(\mathtt{RET\_GUID}) := \top$	22
		else $\mathsf{encGUID} := null$	23
	$\xleftarrow{\mathsf{blindID}, \mathrm{n}_{\mathcal{C}}, \mathsf{authcrypt}, \mathsf{encGUID}, \mathtt{CB}_{\mathcal{C}}}$		
	/* recover card id */		
24	**if** $\neg\mathtt{CB}_{\mathcal{C}}(\mathtt{PB})$		
25	$\mathrm{cert}^*_{\mathcal{C}} := \mathsf{blindID}$		
26	$\mathsf{ID}_{\mathcal{C}} := \mathrm{HASH}(\mathrm{cert}^*_{\mathcal{C}})_{\vert 1..8}$		
27	**else** $\mathsf{ID}_{\mathcal{C}} := \mathsf{blindID}$		
	/* if card used PB and terminal can't find card in registry, restart protocol */		
28	**if** $\neg$PB.contains($\mathsf{ID}_{\mathcal{C}}$) **AND** $\mathtt{CB}_{\mathcal{C}}(\mathtt{PB})$		
29	delete $\mathsf{esk}_{\mathcal{T}}$		
30	restart O-ZKM with $\mathtt{CB}_{\mathcal{T}}(\mathtt{PB}) = \bot$		
31	**if** $\neg\mathtt{CB}_{\mathcal{C}}(\mathtt{PB})$		
32	extract $\mathsf{pk}_{\mathcal{C}}$ from $\mathrm{cert}^*_{\mathcal{C}}$		
33	validate $\mathsf{pk}_{\mathcal{C}}$ belongs to domain of $\mathscr{E}$		
34	$\mathsf{Z} \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{esk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{C}})$		
35	delete $\mathsf{esk}_{\mathcal{T}}$		
36	**else** obtain Z from PB registry (key $\mathsf{ID}_{\mathcal{C}}$)		
	/* compute session keys */		
37	$(\mathsf{sk}_{\mathsf{cfrm}}, \mathsf{sk}_{\mathsf{MAC}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{RMAC}}, \mathsf{nextZ})$ $\leftarrow \mathsf{KDF}(\mathsf{Z}, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{C}}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}\vert 1..16}, \mathrm{n}_{\mathcal{C}}))$		
38	delete Z		
39	check $\mathsf{authcrypt} = \mathsf{CMAC}_{\mathsf{sk}_{\mathsf{cfrm}}}(\mathtt{"KC\_1\_V"} \Vert \mathsf{ID}_{\mathcal{C}} \Vert \mathsf{ID}_{\mathcal{T}} \Vert \mathsf{epk}_{\mathcal{T}\vert 1..16})$		
40	delete $\mathsf{sk}_{\mathsf{cfrm}}$		
41	**if** $\mathcal{C}$.supports(PB)		
42	update PB registry for $\mathcal{C}$: register nextZ under $\mathsf{ID}_{\mathcal{C}}$		
43	**if** $\mathtt{CB}_{\mathcal{C}}(\mathtt{RET\_GUID})$		
44	$\mathsf{GUID} := \mathsf{encGUID} \oplus \mathsf{AES}_{\mathsf{sk}_{\mathsf{Enc}}}(IV)$		
45	**else** extract GUID from $\mathrm{cert}_{\mathcal{C}}$		
46	**if** $\mathsf{C.Vrf}(\mathrm{cert}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{CA}}) = 0$ **abort**		

Figure 3: OPACITY with Zero Key Management

Table 3: Steps of Card in FS-protocol

Line	Step	Description
	$\xrightarrow{\mathrm{cert}_{\mathcal{T}},\mathsf{epk}_{\mathcal{T}},\mathtt{CB}_{\mathcal{T}}}$	Receive terminal's certificate, ephemeral public key and control byte $\mathtt{CB}_{\mathcal{T}}$. Control byte defines whether or not card should use persistent binding.
2	**if** $\mathsf{C.Vrf}(\mathrm{cert}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{CA}}) = 0$ **abort**	Validate the terminal's certificate and abort if verification fails.
3	extract $\mathsf{ID}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{T}}$ from $\mathrm{cert}_{\mathcal{T}}$	Extract public key from certificate.
4	initialize $\mathtt{CB}_{\mathcal{C}}$ according to O-FS mode	Initialize control byte $\mathtt{CB}_{\mathcal{T}}$ indicating that the FS-protocol is to be used and specifying whether card supports persistent binding.
5	look for $\mathsf{ID}_{\mathcal{T}}$ in PB registry	If card supports persistent binding, lookup terminal in registry.
6	**if** $\neg\mathtt{CB}_{\mathcal{T}}(\mathtt{PB})$ **OR** $\neg\mathcal{C}$.supports(PB) **OR** $\neg$PB.contains($\mathsf{ID}_{\mathcal{T}}$)	If persistent binding is not to be used (card or terminal does not support PB or terminal cannot be found in registry), then
7	validate $\mathsf{pk}_{\mathcal{T}}$ belongs to domain of $\mathscr{E}$	Validate terminal's public key with respect to elliptic curve $\mathscr{E}$.
8	$(\mathsf{esk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$	Generate ephemeral elliptic curve key pair.
9	$\mathsf{Z}_1 \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{esk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{T}})$	Compute intermediate value Z_1 using Diffie–Hellman function $\mathcal{DH}_{\mathscr{E}}$ with terminal's public and card's private ephemeral key.
10	$(k_1, k_2) \leftarrow \mathsf{KDF}\,(\mathsf{Z}_1, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}))$	Generate temporary keys k_1 and k_2.
11	$\mathsf{OpaqueData} \leftarrow \mathsf{AES}_{k_1}(\mathrm{cert}_{\mathcal{C}})$	Encrypt card certificate under key k_1 and store result in variable OpaqueData.
12	$\mathsf{otID} := \mathsf{epk}_{\mathcal{C}}$	Store card's ephemeral public in variable otID.
13	$\mathsf{Z} \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{sk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{T}})$	Compute Z using Diffie–Hellman function on card's private and terminal's public ephemeral keys.
14	delete temporary keys Z_1, k_1	Delete temporary keys.
15	**else**	Else, if persistent binding is to be used.
16	obtain $\mathsf{Z}, \mathsf{otID}$ from PB registry (key $\mathsf{ID}_{\mathcal{T}}$)	Lookup values Z and otID in persistend binding registry under key $\mathsf{ID}_{\mathcal{T}}$ (the terminal's id).
17	generate nonce $\mathrm{n}_{\mathcal{C}}$	Generate a nonce.
18	$\mathsf{OpaqueData} := \mathrm{n}_{\mathcal{C}}$	Store nonce in variable OpaqueData.
19	$k_2 := \mathrm{n}_{\mathcal{C}}$	Store nonce in variable k_2.
20	update control byte: $\mathtt{CB}_{\mathcal{C}}(\mathtt{PB}) := \top$	Update the card's control byte to indicate that persistent binding was used.
21	$(\mathsf{sk}_{\mathsf{cfrm}}, \mathsf{sk}_{\mathsf{MAC}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{RMAC}}, \mathsf{nextOtID}, \mathsf{nextZ}) \leftarrow \mathsf{KDF}(\mathsf{Z}, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{otID}_{\lvert 1..8}, \mathsf{epk}_{\mathcal{T}\lvert 1..16}, k_2))$	Comptue session keys and values nextOtID and nextZ which are needed for future persistent binding sessions.
22	delete temporary keys $\mathsf{Z}, k_2, \mathsf{esk}_{\mathcal{C}}, \mathsf{epk}_{\mathcal{C}}$	Delete remaining temporary keys.
23	$\mathsf{authcrypt} \leftarrow \mathsf{CMAC}_{\mathsf{sk}_{\mathsf{cfrm}}}(\mathtt{"KC\_1\_V"} \| \mathsf{otID}_{\lvert 1..8} \| \mathsf{ID}_{\mathcal{T}} \| \mathsf{epk}_{\mathcal{T}\lvert 1..16})$	Compute a MAC over the first 8 bytes of otID (card's public ephemeral key or value from PB registry), the terminal's id and the first 16 bytes of the terminal's public ephemeral key.
24	delete $\mathsf{sk}_{\mathsf{cfrm}}$	Delete key $\mathsf{sk}_{\mathsf{cfrm}}$ that was used for computing the MAC in previous line.
25	**if** $\mathcal{C}$.supports(PB) **AND** $\mathtt{CB}_{\mathcal{T}}(\mathtt{PB})$	If card supports and terminal requests persistent binding, then
26	register $\mathsf{nextZ}, \mathsf{nextOtID}$ for $\mathsf{ID}_{\mathcal{T}}$ in PB registry	Store nextZ, nextOtID under key $\mathsf{ID}_{\mathcal{T}}$ in persistent binding registry.
	$\xleftarrow{\mathsf{OpaqueData},\mathsf{authcrypt},\mathtt{CB}_{\mathcal{C}},\mathsf{otID}}$	Send data to terminal. Value OpaqueData is either the encrypted card certificate or a nonce (if PB was used), authcrypt is a MAC, otID is used for PB or contains card's public ephemeral key, and $\mathtt{CB}_{\mathcal{C}}$ is the cards control byte indicating whether persistent binding is supported and whether it was used.

Table 4: Steps of Terminal in FS-protocol

Line	Step	Description
1	$(\mathsf{esk}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$	Generate elliptic curve ephemeral key pair.
	$\xrightarrow{\mathrm{cert}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}, \mathtt{CB}_{\mathcal{T}}}$	Send certificate, ephemeral public key and control byte $\mathtt{CB}_{\mathcal{T}}$ to card. Control byte defines whether or not card should use persistent binding.
	$\xleftarrow{\mathsf{OpaqueData}, \mathsf{authcrypt}, \mathtt{CB}_{\mathcal{C}}, \mathsf{otID}}$	Receive data from card. Value OpaqueData is either the encrypted card certificate or a nonce (if PB was used), authcryptis a MAC, otID is used for PB or contains card's public ephemeral key, and $\mathtt{CB}_{\mathcal{C}}$ is the cards control byte indicating whether persistent binding is supported and whether it was used.
27	**if** $\neg$PB.contains(otID) **AND** $\mathtt{CB}_{\mathcal{C}}$(PB)	If card used persistent binding, but terminal cannot find card in registry (under key otID), then
28	delete $\mathsf{esk}_{\mathcal{T}}$	Delete private ephemeral key.
29	restart O-FS with $\mathtt{CB}_{\mathcal{T}}(\mathtt{PB}) = \perp$	Restart OPACITY requesting not to use persistent binding.
30	**if** $\neg\mathtt{CB}_{\mathcal{C}}(\mathtt{PB})$	If card did not use persistent binding.
31	$\mathsf{epk}_{\mathcal{C}} := \mathsf{otID}$	Value otID contains the card's public ephemeral key.
32	validate $\mathsf{epk}_{\mathcal{C}}$ belongs to domain of $\mathscr{E}$	Verify card's public key with respect to elliptic curve $\mathscr{E}$.
33	$\mathsf{Z}_1 \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{sk}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}})$	Compute intermediate value Z_1 using the Diffie–Hellman function $\mathcal{DH}_{\mathscr{E}}$.
34	$(k_1, k_2) \leftarrow \mathsf{KDF}(\mathsf{Z}_1, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{C}}))$	Compute keys k_1, k_2 using the key derivation funktion with input Z_1.
35	$\mathrm{cert}_{\mathcal{C}} \leftarrow \mathsf{AES}_{k_1}^{-1}(\mathsf{OpaqueData})$	Decrypt card's certificate with key k_1
36	**if** $\mathsf{C.Vrf}(\mathrm{cert}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{CA}}) = 0$ **abort**	Verify card's certificate and abort if verification fails.
37	extract $\mathsf{pk}_{\mathcal{C}}$ and GUID from $\mathrm{cert}_{\mathcal{C}}$	Extract card's public key and the owner's identity (GUID) from certificate.
38	$\mathsf{Z} \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{esk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{C}})$	Compute intermediate value Z using the Diffie–Hellman function $\mathcal{DH}_{\mathscr{E}}$ with card's public and terminal's private ephemeral keys.
39	delete temporary keys Z_1, k_1	Delete temporary keys, which are no longer needed.
40	**else**	
41	$k_2 :=$ OpaqueData	Set variable k_2 to received value OpaqueData (which should contain the nonce used by the card).
42	obtain Z and GUID from PB registry (key otID)	Lookup values Z and GUID in persistent binding registry which is stored at address otID.
43	$(\mathsf{sk}_{\mathsf{cfrm}}, \mathsf{sk}_{\mathsf{MAC}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{RMAC}}, \mathsf{nextOtID}, \mathsf{nextZ}) \leftarrow \mathsf{KDF}(\mathsf{Z}, \mathsf{len}, , \mathsf{info}(\mathsf{ID}_{\mathcal{T}}, \mathsf{otID}_{\lvert 1..8}, \mathsf{epk}_{\mathcal{T}\lvert 1..16}, k_2))$	Comptue session keys and values nextOtID and nextZ which are needed for future persistent binding sessions.
44	delete keys $\mathsf{Z}, k_2, \mathsf{esk}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}$	
45	check $\mathsf{authcrypt} = \mathsf{CMAC}_{\mathsf{sk}_{\mathsf{cfrm}}}(\mathtt{"KC\_1\_V"} \| \mathsf{otID}_{\lvert 1..8} \| \mathsf{ID}_{\mathcal{T}} \| \mathsf{epk}_{\mathcal{T}\lvert 1..16})$	Verify MAC authcryptand abort if verification fails
46	delete $\mathsf{sk}_{\mathsf{cfrm}}$	
47	**if** $\mathcal{C}$.supports(PB)	If the card supports persistent binding, then
48	update PB registry for $\mathcal{C}$: register nextZ and GUID under nextOtID	Store value nextZ and GUID under key nextOtID in persistent binding registry.

Table 5: Steps of Terminal in ZKM-protocol

Line	Step	Description
1	$(\mathsf{esk}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$	Generate an elliptic curve ephemeral key pair for the terminal.
	$\xrightarrow{\mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}}, \mathtt{CB}_{\mathcal{T}}}$	Send terminal ID, the public key and a control byte indicating whether to use persistent binding and whether to return an encrypted GUID.
	$\xleftarrow{\mathsf{blindID}, \mathrm{n}_{\mathcal{C}}, \mathsf{authcrypt}, \mathsf{encGUID}, \mathtt{CB}_{\mathcal{C}}}$	Card returns value blindID, a nonce n, mac authcrypt, encrypted GUID encGUID (or null), and the card's control byte.
24	**if** $\neg \mathtt{CB}_{\mathcal{C}}(\mathtt{PB})$	If card did not use persistent binding ($\mathtt{CB}_{\mathcal{C}}(\mathtt{PB}) = \perp$)
25	$\quad \mathrm{cert}^{*}_{\mathcal{C}} := \mathsf{blindID}$	Value blindIDcontains blinded certificate (without GUID).
26	$\quad \mathsf{ID}_{\mathcal{C}} := \mathrm{HASH}(\mathrm{cert}^{*}_{\mathcal{C}})_{\mid 1..8}$	Obtain card ID (first 8 bytes of hash of blinded certificate)
27	**else** $\mathsf{ID}_{\mathcal{C}} := \mathsf{blindID}$	Else (if card used PB), value blindIDcontains card's ID.
28	**if** $\neg$PB.contains($\mathsf{ID}_{\mathcal{C}}$) **AND** $\mathtt{CB}_{\mathcal{C}}(\mathtt{PB})$	If PB registry does not contain card $\mathsf{ID}_{\mathcal{C}}$ and card used persistent binding, then
29	$\quad$ delete $\mathsf{esk}_{\mathcal{T}}$	Delete private ephemeral key.
30	$\quad$ restart O-ZKM with $\mathtt{CB}_{\mathcal{T}}(\mathtt{PB}) = \perp$	Restart protocol requesting not to use persistent binding.
31	**if** $\neg \mathtt{CB}_{\mathcal{C}}(\mathtt{PB})$	If card did not use persistent binding, then
32	$\quad$ extract $\mathsf{pk}_{\mathcal{C}}$ from $\mathrm{cert}^{*}_{\mathcal{C}}$	Extract card's public key from certificate.
33	$\quad$ validate $\mathsf{pk}_{\mathcal{C}}$ belongs to domain of $\mathscr{E}$	Validate public key with respect to elliptic curve $\mathscr{E}$.
34	$\quad \mathsf{Z} \leftarrow \mathcal{DH}_{\mathscr{E}}(\mathsf{esk}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{C}})$	Compute intermediate value Z via Diffie–Hellman function $\mathcal{DH}_{\mathscr{E}}$.
35	$\quad$ delete $\mathsf{esk}_{\mathcal{T}}$	Delete private ephemeral key.
36	**else** obtain Z from PB registry (key $\mathsf{ID}_{\mathcal{C}}$)	Else, if card used PB, obtain Z from PB registry.
37	$(\mathsf{sk}_{\mathsf{cfrm}}, \mathsf{sk}_{\mathsf{MAC}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{RMAC}}, \mathsf{nextZ}) \leftarrow \mathsf{KDF}(\mathsf{Z}, \mathsf{len}, \mathsf{info}(\mathsf{ID}_{\mathcal{C}}, \mathsf{ID}_{\mathcal{T}}, \mathsf{epk}_{\mathcal{T}_{\mid 1..16}}, \mathrm{n}_{\mathcal{C}}))$	Compute session keys, the key for validating MAC $\mathsf{authcrypt}(\mathsf{sk}_{\mathsf{cfrm}})$, and 16 byte value nextZ which will later be stored in PB registry.
38	delete Z	Delete intermediate value Z.
39	check $\mathsf{authcrypt} = \mathsf{CMAC}_{\mathsf{sk}_{\mathsf{cfrm}}}(\mathtt{"KC\_1\_V"} \| \mathsf{ID}_{\mathcal{C}} \| \mathsf{ID}_{\mathcal{T}} \| \mathsf{epk}_{\mathcal{T}_{\mid 1..16}})$	Validate message authentication code authcryptand return an authentication error if verification fails.
40	delete $\mathsf{sk}_{\mathsf{cfrm}}$	Delete key for MAC verification.
41	**if** $\mathcal{C}$.supports(PB)	If card supports persistent binding, then
42	$\quad$ update PB registry for $\mathcal{C}$: register nextZ under $\mathsf{ID}_{\mathcal{C}}$	update the PB registry and store value nextZ under key $\mathsf{ID}_{\mathcal{C}}$.
43	**if** $\mathtt{CB}_{\mathcal{C}}(\mathtt{RET\_GUID})$	If the card returned an encrypted GUID, then
44	$\quad \mathsf{GUID} := \mathsf{encGUID} \oplus \mathsf{AES}_{\mathsf{sk}_{\mathsf{Enc}}}(IV)$	Decrypt GUID
45	**else** extract GUID from $\mathrm{cert}_{\mathcal{C}}$	See Section A.1.
46	**if** $\mathsf{C.Vrf}(\mathrm{cert}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{CA}}) = 0$ abort	Validate the certificate. See Section A.1.

Table 6: Steps of Card in ZKM-protocol

Line	Step	Description
	$\xrightarrow{\mathsf{ID}_\mathcal{T}, \mathsf{epk}_\mathcal{T}, \mathtt{CB}_\mathcal{T}}$	Receive terminal ID, the terminals public ephemeral key and a control byte indicating whether to use persistent binding and whether to return an encrypted GUID.
2	$\mathsf{ID}_\mathcal{C} := \mathsf{HASH}(\mathrm{cert}^*_\mathcal{C})_{\mid 1..8}$	Compute card's id: first 8 bytes of hash of blinded certificate (certificate without GUID).
3	initialize $\mathtt{CB}_\mathcal{C}$ according to O-ZKM mode	Initialize card's control byte for the ZKM mode. Control byte contains info that ZKM is to be used and whether card supports persistent binding.
4	look for $\mathsf{ID}_\mathcal{T}$ in PB registry	If card supports persistent binding, lookup terminal in PB registry.
5	**if** $\neg\mathtt{CB}_\mathcal{T}(\mathtt{PB})$ **OR** $\neg\mathcal{C}$.supports(PB) **OR** $\neg$PB.contains($\mathsf{ID}_\mathcal{T}$)	If card and terminal support PB, terminal requested to use PB and terminal is in PB registry, then
6	validate $\mathsf{epk}_\mathcal{T}$ belongs to domain of $\mathscr{E}$	Validate terminal's public key with respect to elliptic curve $\mathscr{E}$.
7	$\mathsf{Z} \leftarrow \mathcal{DH}_\mathscr{E}(\mathsf{sk}_\mathcal{C}, \mathsf{epk}_\mathcal{T})$	Computed intermediate value Z using Diffie–Hellman function $\mathcal{DH}_\mathscr{E}$.
8	$\mathsf{blindID} := \mathrm{cert}^*_\mathcal{C}$ /* see Section 3.1 */	Store blinded certificate in variable blindID. This line is ambigously specified. It seems that if RET_GUID is not set, that then certificate is send in the clear. Also see Section 3.1 and line 45 in the terminal steps.
9	**else**	Else, if persistent binding is to be used, then
10	obtain Z from PB registry (key $\mathsf{ID}_\mathcal{T}$)	Get intermediate value Z from persistent binding registry.
11	$\mathsf{blindID} := \mathsf{ID}_\mathcal{C}$	Set variable blindIDto card's id.
12	update control byte: $\mathtt{CB}_\mathcal{C}(\mathtt{PB}) := \top$	Update the control byte to indicate that persistent binding was used.
13	generate nonce $\mathrm{n}_\mathcal{C}$	Generate 16 byte nonce.
14	$(\mathsf{sk}_\mathsf{cfrm}, \mathsf{sk}_\mathsf{MAC}, \mathsf{sk}_\mathsf{Enc}, \mathsf{sk}_\mathsf{RMAC}, \mathsf{nextZ}) \leftarrow \mathsf{KDF}(\mathsf{Z}, \mathsf{len}, \mathsf{info}(\mathsf{ID}_\mathcal{C}, \mathsf{ID}_\mathcal{T}, \mathsf{epk}_{\mathcal{T}\mid 1..16}, \mathrm{n}_\mathcal{C}))$	Compute session keys, the key for validating MAC $\mathsf{authcrypt}(\mathsf{sk}_\mathsf{cfrm})$, and 16 byte value nextZ which will later be stored in PB registry.
15	delete Z	Delete intermediate value Z.
16	$\mathsf{authcrypt} \leftarrow \mathsf{CMAC}_{\mathsf{sk}_\mathsf{cfrm}}(\mathtt{"KC\_1\_V"} \| \mathsf{ID}_\mathcal{C} \| \mathsf{ID}_\mathcal{T} \| \mathsf{epk}_{\mathcal{T}\mid 1..16})$	Compute MAC on ids of card and terminal and the first 16 bytes of terminal's public key using key $\mathsf{sk}_\mathsf{cfrm}$.
17	delete $\mathsf{sk}_\mathsf{cfrm}$	Delete key $\mathsf{sk}_\mathsf{cfrm}$.
18	**if** $\mathcal{C}$.supports(PB) **AND** $\mathtt{CB}_\mathcal{T}(\mathtt{PB})$	If card supports PB and terminal requests to use it, then
19	register nextZ for $\mathsf{ID}_\mathcal{T}$ in PB registry	Store value nextZ in persistent binding registry under the terminal's id.
20	**if** $\mathtt{CB}_\mathcal{T}(\mathtt{RET\_GUID})$	If terminal requests an encrypted GUID, then
21	$\mathsf{encGUID} := \mathsf{GUID} \oplus \mathsf{AES}_{\mathsf{sk}_\mathsf{Enc}}(IV)$	Encrypt GUID using key sk_Enc and store it in variabel encGUID.
22	update control byte: $\mathtt{CB}_\mathcal{C}(\mathtt{RET\_GUID}) := \top$	Update control byte indicating that GUID was encrypted.
23	**else** $\mathsf{encGUID} := null$	Else set variable encGUID to null.
	$\xleftarrow{\mathsf{blindID}, \mathrm{n}_\mathcal{C}, \mathsf{authcrypt}, \mathsf{encGUID}, \mathtt{CB}_\mathcal{C}}$	Return value blindID(blinded certificate or card's ID), nonce n, mac authcrypt, encrypted GUID encGUID (or null), and the card's control byte.