An Analysis of the EMV Channel Establishment Protocol

C. Brzuska[1], N.P. Smart[2], B. Warinschi[2], and G.J. Watson[2]

[1] School of Computer Science, School of Engineering
Tel Aviv University, Israel.
[2] Dept. Computer Science,
University of Bristol, UK.

Abstract. With over 1.6 billion debit and credit cards in use worldwide, the EMV system (a.k.a. "Chip-and-PIN") has become one of the most important deployed cryptographic protocol suites. Recently, the EMV consortium has decided to upgrade the existing RSA based system with a new system relying on Elliptic Curve Cryptography (ECC). One of the central components of the new system is a protocol that enables a card to establish a secure channel with a card reader. In this paper we provide a security analysis of the proposed protocol, we propose minor changes/clarifications to the "Request for Comments" issued in Nov 2012, and demonstrate that the resulting protocol meets the intended security goals.

The structure of the protocol is one commonly encountered in practice: first run a key-exchange to establish a shared key (which performs authentication and key confirmation), only then use the channel to exchange application messages. Although common in practice, this structure takes the protocol out of the reach of most standard security models for key-exchange. Unfortunately, the only models that can cope with the above structure suffer from some drawbacks that make them unsuitable for our analysis. Our second contribution is to provide new security models for channel establishment protocols. Our models have a more inclusive syntax, are quite general, deal with a realistic notion of authentication (one-sided authentication as required by EMV), and do not suffer from the drawbacks that we identify in prior models.

1 Introduction

The EMV chip-and-pin system is used to secure a vast number of the world's credit card and ATM transactions and protects electronic banking in many countries [14]. The USA is due to switch from magnetic stripe to EMV cards from October 2015; after this time if a terminal (or ATM) does not support EMV then the merchant (ATM owner) will be liable for any fraudulent transactions and not the card issuer. The current EMV system uses RSA public-key cryptography, combined with DES and AES based symmetric-key cryptography. In the system, bank or credit card customers receive a plastic card containing an embedded chip that holds several cryptographic keys and can perform various cryptographic operations. The card is used to communicate with a terminal (typically a point-of-sale terminal in a shop, but other terminals are possible). In addition the card can produce cryptographically secured data which is sent to the cardholder and merchant's banks for processing. Nonetheless, the cryptographic functionality provided by the card in its first generation incarnation is relatively limited.

As part of a major reworking of the system, the EMV consortium has decided to replace RSA with ECC based systems and to let the card provide additional cryptographic functionalities. In November 2012 EMVCo released a Request-For-Comments [15] on a draft specification for an important sub-protocol within the system; namely a protocol that allows a card to establish a channel with a terminal. Calling the security of these protocols "important" is a serious understatement: the total number of public keys and certificates (1.62 billion [14]) deployed in the EMV systems dwarfs the paltry 5.8 million TLS certificates found in [10].

The problem of establishing and implementing secure channels is central to practical uses of cryptography and a superficial look at existing literature would lead one to believe that this is a solved problem. What can be simpler than first running a secure key-exchange protocol and then using the resulting keys to encrypt and authenticate the messages to be sent?

Indeed, there is a plethora of works looking at key establishment [4, 5, 8] and a similar number of works treating secure channels based on shared keys [1, 6, 17]. Unfortunately, traditional key agreement models such as those following the schema set out by Bellare and Rogaway [4] have been shown to be less useful to analyze

deployed protocols. In particular, they demand for session keys to be indistinguishable from a random string. However, whenever practical protocols implement explicit key confirmation, then key indistinguishability is violated.

Realizing that real-world protocols, such as TLS, are therefore outside the reach of the traditional models for key-exchange and channels, has triggered renewed interest in formal models for secure channels [16, 12, 7]. These approaches deal with what is essentially an overlap between the key-exchange part and the secure channel part of a channel-establishment protocol by either modifying the protocol, analyzing the overall protocol monolithically, or developing methods that allow for a modular analysis despite the overlapping phases.

The structure of the EMV protocol for establishing channels follows the recipe described above: during the key-exchange phase itself, the channel is already used before the deployed keys are accepted; and the messages that are sent over the channel are crucial for the security of the overall protocol. Our work can therefore be seen as a continuation of the recent thrust on research on models for channel establishment protocols. Below we describe the state-of-the art for such models, identify some of their weaknesses, and overview our results.

EXISTING MODELS FOR SECURE CHANNEL ESTABLISHMENT. When studying the combined properties of a key establishment and a secure channel protocol, one usually considers a security game where the execution model and the attack interface for key exchange and secure channels are "glued" together; and the adversary tries to break the secure channel using additional information that possibly leaked during the key-exchange phase. In particular, such a model also allows to analyze protocols that do not clearly separate the two components.

There are currently two approaches to prove the security in such a model. Very roughly, the first approach is to relax the security requirement on the keys by demanding that they are sufficiently strong to be used for the primitives that make-up the channel, and then show that the channel security relies only on these primitives. This *modular approach* is explored in [7], where a game-based composition theorem shows that a key-exchange protocol that is sufficiently strong for the primitives and a channel whose security reduces to these primitives automatically yield a secure protocol in the composed security model. The approach is shown to work for real-world protocols such as TLS.

In this paper, we prefer to avoid the machinery needed to work within this framework and instead concentrate on the approach of Jager et al. [12]. They propose to analyze channel establishment protocols, monolithically, with respect to a security model devised for this specific task. The models that they give are tailored for TLS and are not immediately applicable for EMV. Worse, both the original version of the model [12] and a more recent refined version [11] do not seem to appropriately capture the level of security that one would like. In brief, the former model is too strong, to the point that it actually rules as insecure protocols like TLS and the one that we analyze in this paper. The refined version, on the other hand seems to be too weak, as it takes away one of the adversary's abilities, an ability that reflects possible real-world powers.

A bit more in detail, the issue concerns the ability of an adversary to "reveal" a key, and its interaction with how "partnering" is defined. We explain these concepts next. Traditionally, reveal queries model the unintended leakage of session keys from a participant to assure that keys which leak from one session should not affect the security of other sessions. To distinguish between the same session and "other sessions", key exchange protocols match each session of a protocol with a single session of the intended partner via a partnering mechanism [3]. One of the first formulations of partnering relies on "matching" conversations [4] (the outgoing/incoming messages of one sessions are the same as the incoming/outgoing messages of its partner). The requirement is that if a session accepts, then it has had a matching conversation with "the right" partner. Unfortunately, in any protocol where some messages are sent encrypted with the key that is derived, the above requirement cannot hold. An attacker can proceed as follows. When an encrypted message goes on the network, block it, reveal the key that was used for encryption and then send to the recipient a different encryption of the same message, deploying fresh randomness. The two partners will not have a matching conversation, although the protocol will be executed successfully. In Appendix C.2 we describe an attack against entity authentication

via matching conversations for TLS when an adversary is permitted to reveal keys as soon as they are derived. We stress that our attack uncovers subtleties in modelling and is not an actual attack on the TLS protocol.

One approach to circumvent this attack is to preclude the adversary from performing such a reveal. This is the approach taken in [11] which only considers that keys can be revealed once the session in which they are derived had accepted. We find this restriction unsatisfactory. If reveals are considered possible, then they should be able to target a key as soon as that key has been derived. In particular, revealing a key that has just been used to perform an encryption should be allowed. A more in-depth discussion of weaknesses present in the existent models for channel-establishment protocols is in Appendix C.

Our Contribution

In this paper we present a new definitional framework which addresses the problems identified in previous approaches. In particular we present a security model which is particularly tailored to the case of key-exchange followed by the creation of a secure channel. Our new framework is conceptually simpler than previous models and can be further extended to capture one-sided key agreement followed by composition with a secure channel. Below we highlight some of our contributions and techniques.

MODELS FOR CHANNEL-ESTABLISHMENT PROTOCOLS. For entity authentication, we deal with the realistic case of one-sided authentication. This is demanded by the protocol that we analyze which is inherently one-sided: the card is authenticated, while the terminal is not. We remark that existing models for channel-establishment concentrate on mutual authentication, and the case of one-sided authentication had been considered only sporadically in the key-exchange literature.

We also resolve the issue of bad interaction between partnering and reveal queries. Here, we take a different route than [12, 11]. Instead of weakening the adversary, we relax the partnering requirement: we only demand that partners agree on a common session identifier. This approach originates in the work of Bellare, Pointcheval, and Rogaway [3], is standard in key-exchange literature, and reflects an intuitively appealing level of security.

Finally we model and analyze unlinkability properties of the proposed protocol from EMVCo. One of the design criteria of the protocol is a mild form of unlinkability; an adversary that sees a message flow between a terminal and a card should not be able to link this card's current transaction with a previous transaction from the same card. The protocol aims to ensure this by not transmitting the certificate in the clear, however the proposed protocol also uses a performance optimization in that the card uses a small ephemeral private key. We establish that using a small ephemeral key in this way should be avoided.

PROTOCOL ANALYSIS & EMV RECOMMENDATIONS. The EMV channel establishment protocol consists of a key exchange phase and an application phase. The key exchange phase is an ECC-based Diffie-Hellman-like protocol with one-sided authentication. We analyze the EMV channel establishment protocol and identify the assumptions under which it can be proved secure with respect to the notion that we put forth. We end this introduction by pointing out a number of recommendations related to the EMV protocol which have been passed to the designers as a result of our analysis:

1. The resulting Diffie–Hellman key should be hashed down to obtain the used symmetric keys. The proposal in [15] says to use a hash function or the x-coordinate of the elliptic curve point as the key derivation function. We consider any choice not using a hash function to be insecure; indeed security crucially relies on the hash being applied.
2. The resulting keys should be used in a uni-directional manner; thus two keys need to be obtained from the hashing process. This avoids a large number of potential replay attacks on the application layer. If this was not done, the application layer would need to be implemented extremely carefully to thwart these attacks. Having two keys, one for each direction, makes the design of a secure application layer less vulnerable. We have implicitly assumed, as this is not stated in [15], that the resulting secure channel should be secure against adversaries both deleting messages and playing messages out of order; since this is the usual definition of a secure channel.

3. The card ephemeral key a should *not* be selected from the set $\{0,1\}^{32}$. We suggest that it is not restricted in size and instead chosen at random from $\mathbb{F}_q$. If the value a is selected from the set $\{0,1\}^{32}$, then this has a significant effect on security. Not only does it reduce the scheme's ability to achieve unlinkablity. But in addition, when a is selected from a small set an adversary could establish two sessions of one card which share the same key with a single terminal.

2 Scheme

Our presentation follows that in [15], augmented with information obtained from public discussions with the authors of the protocol at several meetings. The basic underlying idea of the protocol is to use a Diffie–Hellman key exchange in which one side (the card) has a static public key. In order to achieve unlinkability the certificate of this public key is not passed in the clear; instead, the card's static Diffie–Hellman key share is randomized by an additional ephemeral secret. The resulting Diffie–Hellman key is then hashed using a cryptographic hash function; which we will model as a random oracle.

The protocol uses a Diffie–Hellman group defined over an elliptic curve $G = E(\mathbb{F}_p)$ having group order a prime q and also uses a base point $P \in G$. The prime q is a function of an implicit security parameter k, but in practice the group is fixed and so all our results are given in the concrete security setting.

After the protocol has established secret keys, it uses them in a secure channel protocol $(\mathsf{SendCh}, \mathsf{ReceiveCh})$. On input an application message m and state st_e, SendCh returns a channel message $\mathfrak{ch}$. On input a channel message $\mathfrak{ch}$ and state st_d, $\mathsf{ReceiveCh}$ returns an application message m. The secure channel protocol is based on a stateful authenticated-encryption (AE) scheme $\mathsf{AE} = \{\mathsf{enc}, \mathsf{dec}\}$. We assume that all plaintext headers used by the secure channel are unauthenticated, implying that no header is sent in clear as part of the AE scheme. The states st_e and st_d here model the fact that in practice sequence numbers are used to ensure that messages are delivered in order, thus the operations are stateful. We assume that the underlying AE scheme satisfies the properties of indistinguishability under chosen-ciphertext attack (IND-sfCCA) and integrity of plaintexts (INT-sfPTXT) for such stateful schemes, assuming the key-agreement scheme has generated a randomly distributed key. See Appendix A for formal definitions of these security notions.

We also assume that there is a public-key signature algorithm used to define certificates. In particular each card C has a long term public/private key pair (Q_C, d), where $d \in \mathbb{F}_q$ and $Q_C = dP \in G$. A certificate is a signature/message pair $\mathrm{cert}_C = (\mathsf{sig}_{\mathsf{sk}}(Q_C), Q_C)$ provided by an issuing authority with a key pair $(\mathsf{pk}, \mathsf{sk})$ for some (unspecified) public-key signature algorithm $(\mathsf{sig}, \mathsf{ver})$. All that we require of the signature algorithm is that it be existentially unforgeable under a chosen-message attack (EUF-CMA). Again Appendix A gives the precise security definition we will use.

We are now in a position to define the EMV key establishment and secure channel protocol in Figure 1. As well as the components above the protocol makes use of a hash function H which takes elements in the group G and maps them onto a pair of keys for the AE scheme. The keys are used to secure the communication in both directions; we propose the use of two keys so that replay attacks are prevented at the level of the protocol as opposed to needing to be dealt with at the application layer.

As mentioned in the introduction the proposal by EMVCo [15] suggest that the ephemeral secret[3] a should be small, (less than 2^{32}). They state that this choice is "set to be fit for purpose for blinding a one-off session key". First note that the unlinkability property may be hard to achieve when a is small: Given two public keys Q_1 and Q_2 and the first message of a session aQ_i, there is an obvious square-root attack which determines Q_i when a is small, i.e. an attack which runs in time roughly 2^{16} operations.

More seriously, the security of entity authentication would also be at risk. An adversary can perform an attack which allows two sessions of a card to share the same key with a single terminal. Consider a card with public key $Q = dP$ and assume that the ephemeral blinding exponents a are small. In the first session the card chooses a_1 at random and sends $a_1 Q$. The terminal will respond by choosing e at random and returning

[3] In the EMV draft a is denoted r.

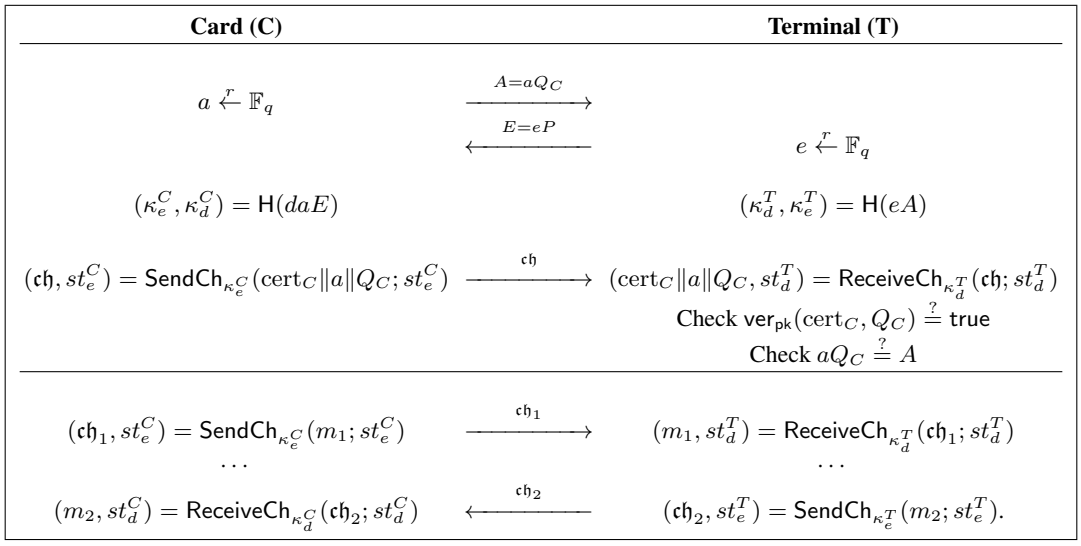

Fig. 1. Combined Authenticated Key Agreement Scheme and Secure Channel Protocol. Note that $\kappa_e^C = \kappa_d^T$ and $\kappa_d^C = \kappa_e^T$.

$E = eP$. Their established key for this session will then be $\kappa = a_1 deP$. Next the card wants to establish a second session. It chooses at random a new ephemeral blind a_2 and sends $a_2 Q$. When the ephemeral blinds (a_1 and a_2) are small then an adversary can easily find $a' = a_1/a_2$. The adversary can then calculate $a'E$ and return this to the card. This second card session now establishes the key $a_2 d(a'E) = a_2 d(a_1/a_2)eP = a_1 deP = \kappa$; the same key as the first session. As a result of this (depending on the setting) the adversary may be able to perform a replay attack on the second card session.

There are other approaches which could prevent the latter attack (in cases where unlinkability is not an issue) but we believe increasing the size of a to be the simplest and to offer the least chance of implementation errors being introduced. In the rest of the paper we assume that a is chosen from $\mathbb{F}_q$ and therefore our security results apply only to this case.

3 New Security Models

In this section we present our security models for the secure channel establishment and unlinkability. Most of the section is however devoted to the more complex case of modeling secure channel establishment.

PRELIMINARIES. Before giving our new definition we present some preliminary definitions. Let I be the set of participants. Each participant has a distinct ID i, long-term public key pk_i and corresponding secret key sk_i. The protocol description is defined by two efficiently computable stateful (sub)-protocols $P = \{\Pi, \mathcal{G}\}$. The protocol Π defines how honest parties behave and $\mathcal{G}$ is a public/private key pair generation algorithm. Each execution of this algorithm maintains the following state information:

- $st_k \in \{0,1\}^*$ is some state for the key exchange.
- $\delta \in \{\texttt{derived}, \texttt{accept}, \texttt{reject}, \perp\}$ is current state of the key exchange (initialised to $\perp$).
- $\rho \in \{\texttt{initiator}, \texttt{responder}\}$ is the role of the participant.
- sid a session identifier.
- pid a partner identifier
- $\kappa = (\kappa_e^\rho, \kappa_d^\rho) \in (\{0,1\}^* \cup \{\perp\})^2$. This is the agreed pair of keys. The order of these keys depends on the role $\rho \in \{\texttt{initiator}, \texttt{responder}\}$ and $\kappa = (\perp, \perp)$ unless $\delta = \texttt{derived}$.

A CLASS OF PROTOCOLS. In this paper we are concerned with the large class of protocols obtained by combining key-exchange with channel protocols, and where the two components cannot be easily disentangled. Existing models are either too specific [12] or do not consider the case of overlap between the components [7].

The syntax that we provide for such protocols naturally implies a classification of the different types of messages that may occur in an execution. The classification is a crucial ingredient of our security definitions. Specifically, we identify three different execution "modes" of protocols: establishing a key, sending, respectively receiving messages from the established channel. Formally, we define the honest operation of a participant by a triple $\Pi = (\mathsf{KeyExch}, \mathsf{SendCh}, \mathsf{ReceiveCh})$.

Some of the messages sent during the key-exchange may travel over the channel so, strictly speaking, KeyExch may make use of the latter algorithms. To facilitate the description of the resulting complex interaction we define the algorithm EstChannel which, essentially, is in charge of establishing the channel. This algorithm may make calls to the algorithms defining Π. As an example, for EMV, one can view EstChannel as everything above the line drawn in Figure 1, see Figure 4 in Section 4 for full details. At any point during its execution protocol Π takes as input a message m and a message type, $\mathtt{type} \in \{\mathtt{app}, \mathtt{ch}\}$ (indicating whether the message was received from the user's application or the channel, respectively), runs the appropriate algorithm, and returns the output of that algorithm.

We now detail the execution of Π which is summarised in Figure 2. Prior to the channel being established (i.e. prior to $\delta = \mathtt{accept}$) whenever the input message m has type ch then EstChannel will be called. This algorithm will make calls to KeyExch in order to establish the keys of the secure channel. The algorithm KeyExch takes as input a message m and a state st_k, and outputs a new message m' and updated state st_k (shared with EstChannel). The state st_k is used to manage the internal state of KeyExch and will contain the session identifier sid, partner identifier pid, the state of keys δ, the established keys κ_e, κ_d, the states of the secure channel st_e, st_d, and any additional information α. Before any keys have been derived ($\delta = \perp$), EstChannel forwards messages directly to KeyExch. Once keys have been derived ($\delta = \mathtt{derived}$) then we enter the key confirmation stage. At this point EstChannel initialises the states st_e, st_d of the secure channel and makes appropriate[4] calls to SendCh and ReceiveCh before and after calling KeyExch. Finally, once KeyExch outputs $\delta = \mathtt{accept}$ the secure channel has been successfully established.

After the channel has been established whenever the input message type is ch then ReceiveCh will be called. This models messages that are received from the channel (for decryption). It takes as input a message m and state st_d and outputs a message m' for output to the user's application.

When the message type is app then SendCh will be called. This models application messages that are input to be sent (encrypted) on the channel. It takes as input a message m and state st_e and outputs a message m' for output to the channel. Note that if keys have not yet been established ($\delta \neq \mathtt{accept}$) then such a call to SendCh will output $\perp$.

Protocol $\Pi(m, \mathtt{type})$:
if $\mathtt{type} = \mathtt{ch} \wedge \delta \neq \mathtt{accept}$ **then**
  $(m'; st_k) \leftarrow \mathsf{EstChannel}(m; st_k)$
else if $\mathtt{type} = \mathtt{ch}$ **then**
  $(m'; st_d) \leftarrow \mathsf{ReceiveCh}_{\kappa_d}(m; st_d)$
else $(\mathtt{type} = \mathtt{app})$
  $(m'; st_e) \leftarrow \mathsf{SendCh}_{\kappa_e}(m; st_e)$
return m'

Fig. 2. Honest Protocol Execution

EXECUTION MODEL. We consider the standard execution model for key exchange protocols where an adversary $\mathcal{A}$, is assumed to control all communication between participating parties i.e. the adversary can intercept all messages sent and inject any message that he wishes. Let Π_i^s denote the oracle modelling participant $i \in I$ engaged in session s of the protocol described above. Each oracle Π_i^s runs the program Π and maintains the states of that program instance. The adversary can make the following queries:

[4] Such calls will be dependent on the specific protocol construction.

- $\mathsf{NewSession}(i, \rho)$: Create a session for user i with role ρ.
- $\mathsf{Send}(\Pi_i^s, m, \mathtt{type})$: Sends a message m to Π_i^s with type, $\mathtt{type}$. As a result Π_i^s will run Π on input $(m, \mathtt{type})$.
- $\mathsf{Reveal}(\Pi_i^s)$: reveals the current session key κ of Π_i^s.
- $\mathsf{Corrupt}(i)$: reveals the long-term private key of i.

PARTNERING AND FRESHNESS. In order to define security for key-exchange protocols it is necessary to define the notion of partnering. Two participants should only establish a shared key if they have been successfully partnered. There are many approaches to defining such a notion. We begin by discussing the concept of *matching conversations*, introduced by Bellare and Rogaway [4] in the context of authenticated key exchange. A participant's conversation can be defined as a transcript of all the messages it receives and sends. As the name suggest, matching conversations defines two participants to be partnered if their transcripts *match*. It is this approach which is followed by Jager et al. [12] in their definition of ACCE. Unfortunately, when protocols use the session key to encrypt messages as part of a key-confirmation step, attacks may be possible which violate the requirements of matching conversations[5] (cf. Appendix C.2). Notice however that while such attacks violate the matching conversation property, they should perhaps not be considered attacks. The plaintext that was sent by one party still reached its intended recipient. We interpret this attack as a limitation of the model: it may rule out as insecure protocols with no obvious weaknesses.

Our formulation is routed in the definition of partnering based on session identifiers by Bellare et al. [3]. Informally, Bellare et al. declared two oracles partnered if they have already derived keys and i) they both share the same session identifier sid, ii) they derived the same key κ, and iii) one oracle is an initiator and the other a responder. Moreover, to ensure each oracle accepted with only a single partner we also ask that iv) there should exist no other oracle which has derived keys and holds the same session identifier. Bellare et al. make a distinction between an oracle accepting and terminating. Accepting defines the event that the session keys have been established but the key confirmation steps are still to follow. An oracle terminates after the key confirmation steps have completed. Once keys are accepted they may be revealed but the key-exchange protocol has yet to terminate. We argue that a key is not "accepted" until after the key confirmation step since this step may fail. As a result, we use the terms derived and accepted, where derived corresponds to Bellare et al.'s accepted and our accepted corresponds to their terminated.

The specific formulation we give is similar but a stronger notion than that of Rogaway and Stegers [19]. First we define a notion of partnering which informally states that two oracles which have derived keys are *partners* if they share the same session identifier. The definition makes use of the following predicate on two oracles Π_i^s and Π_j^t holding $(\kappa_i^s, sid_i^s, pid_i^s)$ and $(\kappa_j^t, sid_j^t, pid_j^t)$, respectively:

$$\mathsf{P}(\Pi_i^s, \Pi_j^t) = \begin{cases} \mathsf{true} & \text{if } sid_i^s = sid_j^t \ \wedge \ \delta_i^s, \delta_j^t \in \{\mathtt{derived}, \mathtt{accept}\} \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Definition 1. *(Partner) Two oracles Π_i^s and Π_j^t are said to be partnered if* $\mathsf{P}(\Pi_i^s, \Pi_j^t) = \mathsf{true}$.

We follow this with three further definitions related to partnering which will be necessary when we give our main security definitions later. It shall be required that partnerings are valid, confirmed and unique. In short these three requirements ensure that any oracle that accepts, has a partner and that this partner is unique. More specifically, *Valid* means that partners will have corresponding partner identifiers (i.e. they both believe they are talking with each other), have different roles and share the same key. *Confirmed* partners ensures that for each oracle that accepts there exists *at least* one partner and for *unique* partners that there exists *at most* one.

[5] The adversary reveals the key and then uses this to re-encrypt the confirmation message with new randomness. The two transcripts now differ for this message.

Definition 2. *(Valid Partners) A protocol $P = \{\Pi, \mathcal{G}\}$ ensures valid partners if the bad event* notval *does not occur, where* notval *is defined as follows:*

$$\exists\, \Pi_i^s, \Pi_j^t \text{ such that}$$
$$\left(\mathsf{P}(\Pi_i^s, \Pi_j^t) = \mathsf{true}\right) \wedge \left(pid_i^s \neq j \ \vee\ pid_j^t \neq i \ \vee\ \rho_i^s = \rho_j^t \ \vee\ \kappa_i^s \neq \kappa_j^t\right).$$

Definition 3. *(Confirmed Partners) A protocol $P = \{\Pi, \mathcal{G}\}$ ensures confirmed partners if the bad event* notconf *does not occur, where* notconf *is defined as follows:*

$$\exists\, \Pi_i^s \text{ such that}$$
$$\delta_i^s = \mathtt{accept} \ \wedge\ i, pid_i^s \text{ not corrupt} \ \wedge \left(\mathsf{P}(\Pi_i^s, \Pi_j^t) = \mathsf{false} \ :\ \forall \Pi_j^t\right).$$

Definition 4. *(Unique Partners) A protocol $P = \{\Pi, \mathcal{G}\}$ ensures unique partners if the bad event* notuni *does not occur, where* notuni *is defined as follows:*

$$\exists \Pi_i^s, \Pi_j^t, \Pi_k^r \text{ such that}$$
$$(j,t) \neq (k,r) \ \wedge\ \mathsf{P}(\Pi_i^s, \Pi_j^t) = \mathsf{true} \ \wedge\ \mathsf{P}(\Pi_i^s, \Pi_k^r) = \mathsf{true}.$$

A concept that plays a central role in defining security in two-party protocols is that of "freshness". Intuitively, an oracle is fresh if it has accepted and an adversary had not "tampered" with it in any way, i.e. the adversary has not revealed or corrupted the oracle or its partner. This can be viewed as being related to the bad event notconf with the extra requirement that the oracle has not been revealed. A notion of freshness is necessary when defining security since the security guarantees are only for such oracles. The next definition formalises the concept. As with partnering we first introduced a predicate, $\mathsf{F}(\Pi_i^s) = \mathsf{true}$, if and only if:

1. $\delta_i^s = \mathtt{accept}$.
2. Oracle Π_i^s has not been revealed and user i is not corrupted.
3. No partner oracle of Π_i^s has been revealed and no parent of such a oracle has been corrupted.

Definition 5. *(Fresh) An oracle Π_i^s is **fresh** if $\mathsf{F}(\Pi_i^s) = \mathsf{true}$.*

3.1 Security Definitions: Two-Sided Authentication Setting

We formulate three levels of security: entity authentication, message authentication and message privacy. The later definitions rely on entity authentication and we start by defining that definition.

ENTITY AUTHENTICATION. We consider that an adversary violates entity authentication if he can get a session to accept even if there is no unique session of its intended partner that has derived the same key. More formally, we wish to verify that none of the bad events notval, notconf, notuni occurs (cf. Definitions 2, 3, 4).

First consider the entity authentication experiment entauth that generates public/private key pairs for each user $i \in I$ (by running $\mathcal{G}$) and returns the public keys to $\mathcal{A}_{\mathsf{ent}}$. The experiment then allows the adversary $\mathcal{A}_{\mathsf{ent}}$ to make the queries $\mathsf{NewSession}(i, \rho)$, $\mathsf{Reveal}(\Pi_i^s)$, $\mathsf{Corrupt}(i)$ as well as $\mathsf{Send}(\Pi_i^s, m, \mathtt{type})$ with operations $\mathtt{type} \in \{\mathtt{app}, \mathtt{ch}\}$. We say that an adversary violates entity authentication (and hence "wins" this experiment) if one of the bad events (notval, notconf, notuni) occurs. The adversary's advantage is defined to be:

$$\mathbf{Adv}_{\Pi}^{\mathsf{entauth}}(\mathcal{A}_{\mathsf{ent}}) = \Pr[\mathsf{notval} \vee \mathsf{notconf} \vee \mathsf{notuni}].$$

Definition 6. *(Entity Authentication (EA)) Protocol $P = \{\Pi, \mathcal{G}\}$ is a (t, ϵ_{EA})-**secure EA protocol** if for all adversaries $\mathcal{A}_{\mathsf{ent}}$ running in time at most t, $\mathbf{Adv}_{\Pi}^{\mathsf{entauth}}(\mathcal{A}_{\mathsf{ent}}) \leq \epsilon_{EA}$.*

To define the security experiments for message authentication and privacy we shall make use of the following notation for lists maintained for each Π_i^s as follows:

– Application messages sent $L_{i,s}^{app|sen}$, i.e. the list of all messages m input to $\mathsf{Send}(\Pi_i^s, m, \mathtt{app})$.
– Channel messages sent $L_{i,s}^{ch|sen}$, i.e. the list of all outputs from $\mathsf{Send}(\Pi_i^s, m, \mathtt{app})$.
– Channel messages received $L_{i,s}^{ch|rec}$, i.e. the list of all messages m input to $\mathsf{Send}(\Pi_i^s, m, \mathtt{ch})$.
– Application messages received $L_{i,s}^{app|rec}$, i.e. the list of all outputs from $\mathsf{Send}(\Pi_i^s, m, \mathtt{ch})$.

MESSAGE AUTHENTICATION. We now turn our attention to message authentication. Here we wish to ensure the integrity and authenticity of all messages sent over the channel. For any two partner oracles Π_i^s and Π_j^t, the oracle Π_i^s should only *successfully* receive messages which were output by Π_j^t and vice versa. In the definition which follows we formalise the intuition above by requiring that for any fresh oracle Π_i^s with unique partner Π_j^t, the following holds $\mathsf{Prefix}(L_{i,s}^{app|rec}, L_{j,t}^{app|sen}) = \mathsf{true}$, where $\mathsf{Prefix}(X, Y)$ is the function which outputs true if X is a prefix of Y (provided not empty) and false otherwise.

Consider the authentication experiment auth that generates public/private key pairs for each user $i \in I$ (by running $\mathcal{G}$) and returns the public keys to $\mathcal{A}$. The adversary is permitted to make the queries $\mathsf{NewSession}(i, \rho)$, $\mathsf{Reveal}(\Pi_i^s)$, $\mathsf{Corrupt}(i)$ as well as $\mathsf{Send}(\Pi_i^s, m, \mathtt{type})$ with message $\mathtt{type} \in \{\mathtt{app}, \mathtt{ch}\}$. On the $\mathsf{Send}(\Pi_i^s, m, \mathtt{type})$ query, the game behaves as in Figure 3(a).

For the session matching, we consider the notion of partnering as specified in Definition 1. The notion of freshness that we use in the following definition is according to Definition 5.

We define the following game $\mathbf{Exec}_{\Pi}^{\mathsf{auth}}(\mathcal{A})$ between an adversary $\mathcal{A}$ and challenger $\mathcal{C}$:

1. The challenger $\mathcal{C}$ generates public/private key pairs for each user $i \in I$ (by running $\mathcal{G}$) and returns the public keys to $\mathcal{A}$.[6]
2. $\mathcal{A}$ is allowed to make as many NewSession, Reveal, Corrupt, Send queries as it likes.
3. The adversary stops with no output.

We say that an adversary $\mathcal{A}$ wins the game if there exists Π_i^s with unique partner Π_j^t such that $\mathsf{F}(\Pi_i^s) = \mathsf{true}$ and the list $L_{i,s}^{app|rec}$ is not a prefix of $L_{j,t}^{app|sen}$.

We define the adversary's advantage as:

$$\mathbf{Adv}_{\Pi}^{\mathsf{auth}}(\mathcal{A}) = \Pr[\exists \Pi_i^s, \Pi_j^t : \mathsf{F}(\Pi_i^s) = \mathsf{true} \ \wedge \ \mathsf{P}(\Pi_i^s, \Pi_j^t) = \mathsf{true} \ \wedge \ \mathsf{Prefix}(L_{i,s}^{app|rec}, L_{j,t}^{app|sen}) = \mathsf{false}].$$

Definition 7. *(Message Authenticity (MA)) Protocol $P = \{\Pi, \mathcal{G}\}$ is a (t, ϵ_{MA})-**secure MA protocol** if for all adversaries $\mathcal{A}_{\mathsf{auth}}$ running in time at most t, $\mathbf{Adv}_{\Pi}^{\mathsf{auth}}(\mathcal{A}_{\mathsf{auth}}) \leq \epsilon_{MA}$.*

$\mathsf{Send}(\Pi_i^s, m, \mathtt{type})$:
$m' \leftarrow \Pi_i^s(m, \mathtt{type})$
if $\delta = \mathtt{accept}$ and $\mathtt{type} = \mathtt{app}$ **then**
  $L_{i,s}^{app|sen} \leftarrow L_{i,s}^{app|sen} \| m$
else if $\delta = \mathtt{accept}$ and $\mathtt{type} = \mathtt{ch}$ **then**
  if $m' \neq \perp$ **then** $L_{i,s}^{app|rec} \leftarrow L_{i,s}^{app|rec} \| m'$
return m'

(a) Send query for auth game.

$\mathsf{SendLR}(\Pi_i^s, m_0, m_1, \mathtt{type})$
if $\delta = \mathtt{accept}$ and $\mathtt{type} = \mathtt{app}$ **then**
  $m' \leftarrow \Pi_i^s(m_{b_i^s}, \mathtt{app})$
  $L_{i,s}^{ch|sen} \leftarrow L_{i,s}^{ch|sen} \| m'$
else if $m_0 \neq m_1$ **then** $m' := \perp$
else
  $m' \leftarrow \Pi_i^s(m_0, \mathtt{type})$
  if $\delta = \mathtt{accept}$ and $\mathtt{type} = \mathtt{ch}$ **then**
    if $m' \neq \perp$ and Π_i^s has a partner Π_j^t **then**
      $L_{i,s}^{ch|rec} \leftarrow L_{i,s}^{ch|rec} \| m_0$
      if $\mathsf{Prefix}(L_{i,s}^{ch|rec}, L_{j,t}^{ch|sen}) = \mathsf{true}$ **then** $m' := \emptyset$
return m'

(b) SendLR query for priv game.

Fig. 3. The Send (resp. SendLR) query for the auth (resp. priv) games

[6] Note that in the scheme considered in this paper, public keys of cards are not actually made public to $\mathcal{A}$ but are sent in encrypted form during the confirmation step.

MESSAGE PRIVACY. Next we consider the notion of message privacy. Our definition follows the standard indistinguishability paradigm. The adversary should not be able to determine which set of message $\{m_{01}, m_{02}, m_{03}, ...\}$ and $\{m_{11}, m_{12}, m_{13}, ...\}$ has been transmitted on the secure channel.

The message privacy experiment priv initializes the states as in the authentication experiment auth, except that each session now also holds a random secret bit b_i^s. As before, the adversary can make the queries $\mathsf{NewSession}(i, \rho)$, $\mathsf{Reveal}(\Pi_i^s)$, $\mathsf{Corrupt}(i)$. In addition, we introduce a left-right version of $\mathsf{Send}(\Pi_i^s, m, \mathtt{type})$ which we use to model message privacy. Specifically, the query $\mathsf{SendLR}(\Pi_i^s, m_0, m_1, \mathtt{type})$ takes as input two messages and returns $\mathsf{Send}(\Pi_i^s, m_{b_i^s}, \mathtt{type})$. When $\mathtt{type} \neq \mathtt{app}$ we require that these two message are equal,

$$\mathsf{SendLR}(\Pi_i^s, m, m, \mathtt{type}) = \mathsf{Send}(\Pi_i^s, m, \mathtt{type}).$$

As before, two sessions are considered partners by Definition 1. On the $\mathsf{SendLR}(\Pi_i^s, m_0, m_1, \mathtt{type})$ query, the game behaves as in Figure 3(b).

We define the following game $\mathbf{Exec}_{\Pi}^{\mathsf{priv}}(\mathcal{A})$ between an adversary $\mathcal{A}$ and challenger $\mathcal{C}$:

1. The challenger $\mathcal{C}$, generates public/private key pairs for each user $i \in I$ (by running $\mathcal{G}$) and returns the public keys to $\mathcal{A}$.[7]
2. Adversary $\mathcal{A}$ is allowed to make as many NewSession, Reveal, Corrupt, SendLR queries as it likes.
3. Finally $\mathcal{A}$ outputs a tuple (i, s, b').

We say the adversary $\mathcal{A}$ wins if its output $b' = b_i^s$ and $\mathsf{F}(\Pi_i^s) = \mathsf{true}$ (and has a unique partner) and the output of $\mathbf{Exec}_{\Pi}^{\mathsf{priv}}(\mathcal{A})$ is set to 1. Otherwise the output is 0. Formally we define the advantage of $\mathcal{A}$ as

$$\mathbf{Adv}_{\Pi}^{\mathsf{priv}}(\mathcal{A}) = |\Pr[\mathbf{Exec}_{\Pi}^{\mathsf{priv}}(\mathcal{A}) = 1] - 1/2|.$$

Definition 8. *(Message Privacy (MP)) A protocol $P = \{\Pi, \mathcal{G}\}$ is a (t, ϵ_{MP})-**secure MP protocol** if for all adversaries $\mathcal{A}_{\mathsf{priv}}$ running in time at most t, $\mathbf{Adv}_{\Pi}^{\mathsf{priv}}(\mathcal{A}_{\mathsf{priv}}) \leq \epsilon_{MP}$.*

We call a channel establishment protocol secure if it satisfies all of the three notions above. We call the resulting notion EAMAP for obvious reasons.

Definition 9. *(EAMAP) A protocol $P = \{\Pi, \mathcal{G}\}$ is a (t, ϵ)-**secure EAMAP protocol** if it is a (t, ϵ)-secure EA protocol, a (t, ϵ)-secure MA protocol and a (t, ϵ)-secure MP protocol.*

Remark 1. Our definitions are with respect to the specific type of protocol construction defined in Figure 2. We note however, that our notions can be extended to more general classes of protocols by simply placing fewer restrictions on the Send queries.

Remark 2. Our mechanism of defining message authentication by requiring that the list of messages received by a party is a prefix of the list of the messages sent by its partner is quite flexible. By appropriately modifying this requirement one can also capture more relaxed notions e.g. where packet dropping or reordering is allowed. Furthermore, we expect that with appropriate restrictions this mechanism can also be adapted to deal with fragmentation. This is a common feature of many secure/authenticated channels in practice and has been formally studied by Boldyreva et al. [6], but is not relevant for EMV.

3.2 Security Definitions: One-Sided Authentication Setting

The above security definitions enforce mutual authentication, yet in many scenarios of practical concern only one party needs to be authenticated. For example, the protocol we consider requires authentication of the credit

[7] Note that in the scheme considered in this paper, public keys of cards are not actually made public to $\mathcal{A}$ but are sent in encrypted form during the confirmation step.

card but does not authenticate the communicating terminal. To model this situation we split our set of participants I in two. Let C be the set of authenticated participants (the cards) and let T by the set of unauthenticated participants (the terminals), where unauthenticated participants do not hold a long-term private/public key pair. This formalisation is the same as that of registered and unregistered users in [16]. We say authenticated participants are always initiators and unauthenticated are always responders. As a result of this change we must alter our previous security definitions for entity authentication, message authentication, message privacy and their combination (EAMAP) to consider a one-sided protocol.

ONE-SIDED ENTITY AUTHENTICATION. In the one-sided setting a terminal $j \in T$ wishes to authenticate a card $i \in C$ and establish a key (additionally a secure channel) with this card. Since all $j \in T$ have no long-term secret then it would always be possible for an adversary to impersonate an unauthenticated participant and establish a session with a real card. We need only aim to ensure that a genuine card session is authenticated to an unauthenticated terminal.

We first describe informally the changes we must make to the original two-sided definitions. Recall the definition of partnering (cf. Definition 1 and the associated Definitions 2, 3 and 4). We are now only concerned with the case where a terminal is partnered correctly, i.e. for every terminal that accepts there exists a card session and this card session is unique. As a result this means we must give new one-sided versions of the definitions for valid partners and confirmed partners. These new definitions now specifically focus on the terminal, ensuring that it believes it is talking to the correct card and it is a responder (in the case of valid), and its partner is not corrupt (in the case of confirmed). Note that we do not adjust the uniqueness definition since we still wish to ensure that both a single card session cannot be partnered with two terminal sessions and a single terminal session cannot be partnered with two card sessions.

Definition 10. *(One-Sided Valid Partners) Protocol $P = \{\Pi, \mathcal{G}\}$ ensures valid partners if the bad event* os-notval *does not occur, where* os-notval *is defined as follows:*

$$\exists\ \Pi_i^s, \Pi_j^t \text{ such that}$$
$$i \in T\ \wedge\ j \in C\ \wedge \left(\mathsf{P}(\Pi_i^s, \Pi_j^t) = \mathsf{true}\right) \wedge \left(pid_i^s \neq j\ \vee\ \rho_i^s \neq \texttt{responder} = \rho_j^t\ \vee\ \kappa_i^s \neq \kappa_j^t\right).$$

Definition 11. *(One-Sided Confirmed Partners) Protocol $P = \{\Pi, \mathcal{G}\}$ ensures one-sided confirmed partners if* os-notconf *does not occur, where* os-notconf *is defined as follows:*

$$\exists\ \Pi_i^s \text{ such that}$$
$$i \in T\ \wedge\ \delta_i^s = \texttt{accept}\ \wedge\ pid_i^s \text{ not corrupt}\ \wedge \left(\mathsf{P}(\Pi_i^s, \Pi_j^t) = \mathsf{false}\ :\ \forall \Pi_j^t\right).$$

We consider an adversary that violates one-sided entity authentication if he can get a terminal session to accept if there is no unique card session that has derived the same key. More formally, we define the os-entauth experiment in a similar fashion to before but now say that an adversary violates one-sided entity authentication (and hence "wins" this experiment) if one of the bad events (os-notval, os-notconf, notuni) occurs. The adversary's advantage is defined to be:

$$\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{A}_{\mathsf{ent}}) = \Pr[\mathsf{os\text{-}notval} \vee \mathsf{os\text{-}notconf} \vee \mathsf{notuni}].$$

Definition 12. *(One-Sided Entity Authentication (OS-EA)) A protocol $P = \{\Pi, \mathcal{G}\}$ is a (t, ϵ_{EA})-**secure OS-EA protocol** if for all adversaries $\mathcal{A}_{\mathsf{ent}}$ running in time at most t, $\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{A}_{\mathsf{ent}}) \leq \epsilon_{EA}$.*

ONE-SIDED MESSAGE AUTHENTICITY AND PRIVACY. In order to adapt the definitions of message authenticity and privacy we must consider a one-sided version of freshness. The reason behind this again being that we wish to discount the trivial attack when the adversary impersonates an unauthenticated terminal $j \in T$. A card oracle is defined to be OS-fresh if it has accepted, has not been revealed or corrupted and it is partnered with a genuine terminal oracle. A terminal oracle is defined to be OS-fresh if it has accepted, has not been revealed and it is partnered with a card oracle that has not been revealed or corrupted. We formalise one-sided freshness as follows. As before we define a predicate, $\mathsf{OSF}(\Pi_i^s) = \mathsf{true}$, if and only if:

1. $\delta_i^s = \texttt{accept}$.
2. Oracle Π_i^s has not been revealed.
3. If $i \in C$ then it is uncorrupted and has a partner Π_j^t, where $j \in T$.
4. If $i \in T$ then Π_i^s has a partner Π_j^t, where $j \in C$.
5. No oracle Π_j^t, such that $\mathsf{P}(\Pi_i^s, \Pi_j^t) = \mathsf{true}$, has been revealed and if $j \in C$ then it is uncorrupted.

Definition 13. *(One-Sided Fresh) An oracle Π_i^s, is **OS-fresh** if $\mathsf{OSF}(\Pi_i^s) = \mathsf{true}$.*

Using the above we can alter our previous experiments of auth and priv by requiring that the winning oracle is OS-fresh. We therefore obtain one-sided versions os-auth and os-priv, respectively.

Definition 14. *(OS-MA/OS-MP) A protocol $P = \{\Pi, \mathcal{G}\}$ is a (t, ϵ)-**secure OS-MA protocol** (or OS-MP resp.) if for all adversaries $\mathcal{A}$ running in time at most t, $\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}auth}}(\mathcal{A}) \leq \epsilon$ (or $\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}priv}}(\mathcal{A}) \leq \epsilon$ resp.).*

We call a channel establishment protocol with one-sided authentication secure if it satisfies all three of the notions above.

Definition 15. *(OS-EAMAP) A protocol $P = \{\Pi, \mathcal{G}\}$ is a (t, ϵ)-**secure OS-EAMAP protocol** if it is a (t, ϵ)-secure OS-EA protocol, a (t, ϵ)-secure OS-MA protocol and a (t, ϵ)-secure OS-MP protocol.*

3.3 Security Definitions: Unlinkability

A further property that the EMVCo protocol aims to achieve is unlinkability. This means that it should be hard for an adversary to determine when two particular sessions involve the same card. Goldberg et al. [9] define a related notion of anonymity and unlinkability. They aim to prove a scheme secure if an authenticated party remains anonymous to its unauthenticated partner and hence call this *internal anonymity*. Here we are concerned with eavesdroppers external to the execution and hence define a new notion for *external unlinkability*.

We define this security property in terms of a game between adversary $\mathcal{A}$ and challenger $\mathcal{C}$, $\mathbf{Exec}_{\Pi}^{\mathsf{unlink}}(\mathcal{A})$. Informally, the adversary is able to interact with the card and terminal much as in the key agreement game. At some point the adversary halts the first part of his game, and outputs two card identities on which it wishes to be challenged. The challenger then picks one of these two identities and passes to the adversary new oracles (i.e. card/terminal session) with respect to the chosen identity. The adversary can then make additional queries, bar Reveal or Corrupt queries on the two test oracles. At the end of the experiment the adversary needs to output which identity the challenger selected. More formally the game is defined as follows:

1. The challenger $\mathcal{C}$, generates public/private key pairs for each user $i \in C$ (by running $\mathcal{G}$) and returns the public keys to $\mathcal{A}$.
2. $\mathcal{A}$ is allowed to make as many NewSession, Reveal, Corrupt, Send queries as it likes.
3. At some point $\mathcal{A}$ outputs two identities $i_0 \in C$ and $i_1 \in C$.
4. The challenger then chooses a bit $b \xleftarrow{r} \{0, 1\}$ and creates new oracles $\mathcal{O}_C = \Pi_{i_b}^s$ and $\mathcal{O}_T = \Pi_{j,}^t$ (for some $j \in T$), by calling NewSession.
5. $\mathcal{A}$ continues making queries NewSession, Reveal, Corrupt, Send. However, $\mathcal{A}$ is allowed to query oracles $\mathcal{O}_C$ and $\mathcal{O}_T$ only with the Send query.
6. Eventually $\mathcal{A}$ stops and outputs a bit b'.

We say the adversary $\mathcal{A}$ wins if its output $b' = b$ and $\mathsf{P}(\mathcal{O}_C, \mathcal{O}_T) = \mathsf{true}$. In this case the output of $\mathbf{Exec}_{\Pi}^{\mathsf{unlink}}(\mathcal{A})$ is set to one, otherwise the output is zero. We define the advantage of $\mathcal{A}$ to be

$$\mathbf{Adv}_{\Pi}^{\mathsf{unlink}}(\mathcal{A}) = |\Pr[\mathbf{Exec}_{\Pi}^{\mathsf{unlink}}(\mathcal{A}) = 1] - 1/2|.$$

Definition 16. *(Unlinkability) A protocol $(\Pi, \mathcal{G})$ is $(t, \epsilon_{\mathsf{unlink}})$-unlinkable, if for all adversaries $\mathcal{A}$ running in time t, $\mathbf{Adv}_{\Pi}^{\mathsf{unlink}}(\mathcal{A}) \leq \epsilon_{\mathsf{unlink}}$.*

4 Security Analysis

In this section we state our main security results, and in particular clarify the assumptions under which the proposed EMV channel-establishment protocol is secure, namely the security of the signature scheme that is used to produce the certificates and, the Gap-DH and CDH assumptions in the group that underlies the scheme. We provide formal definitions of these assumptions later in this section.

Before stating our main security results we first give a formal description of the EMV protocol $P = \{\Pi, \mathcal{G}\}$. Here $\mathcal{G}$ is the key-generation algorithm of the underlying signature scheme. As described at the start of Section 3, we define Π in terms of a key-exchange protocol KeyExch and two algorithms which define the secure channel SendCh, ReceiveCh. In order to define channel establishment we utilise an additional algorithm EstChannel which makes calls to the underlying key exchange protocol KeyExch and secure channel algorithms SendCh, ReceiveCh. For simplicity we do not give a separate description of KeyExch but instead define EstChannel directly. We require three further algorithms which are used during the execution, loadkey, keyid, initial. An execution of a card oracle will involve a call to loadkey which will retrieve the public key Q, secret key d and certificate cert associated to an identity i. Recall that $Q = dP$ and $\mathrm{cert} = (\mathsf{sig}_{\mathsf{sk}}(Q), Q)$. An execution of a terminal oracle will involve a call to keyid which will retrieve the identity j associated to a public key Q. The algorithm initial will initialise the states of the secure channel st_e, st_d.

In Figure 4 we define the execution of EstChannel for a session s of a party i with state $st_k = (\alpha, \delta, \rho, st_e, st_d, \kappa_e, \kappa_d, pid, sid)$ where each variable is initiated as $\perp$, the value α is used to store a and A for the card and the terminal respectively.

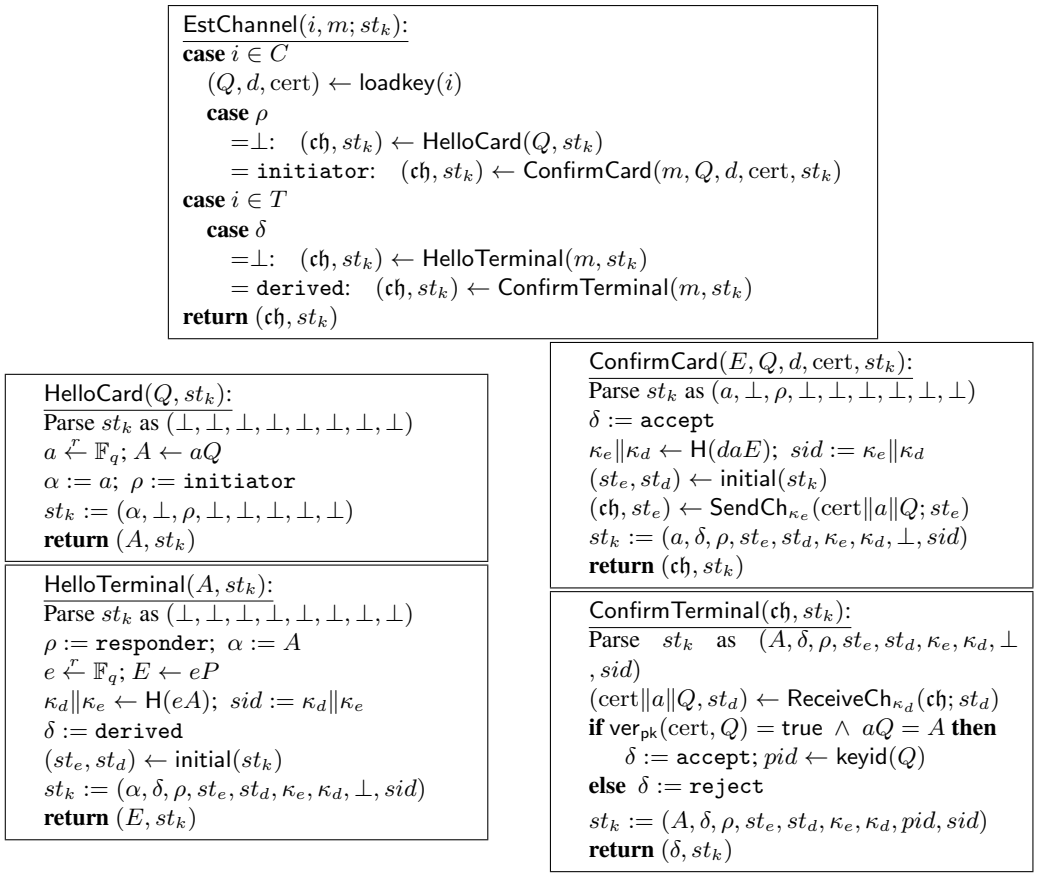

Fig. 4. Channel Establishment Protocol EstChannel for EMV.

We now state our main security result.

Theorem 1. *If the Gap-DH problem is hard, the CDH problem is hard,* $\mathsf{AE} = (\mathsf{enc}, \mathsf{dec})$ *is an ind-sfcca secure and int-sfptxt secure AE scheme, and the signature scheme* $(\mathsf{sig}, \mathsf{ver})$ *used to produce card certificates is EUF-CMA, then the EMV protocol* $P = \{\Pi, \mathcal{G}\}$ *in Figures 1 and 4 is secure in the sense of OS-EAMAP. In particular we have*

- *If there exists an adversary* $\mathcal{A}$ *running in time at most* t *against the entity authentication property of OS-EAMAP security then there are adversaries* $\mathcal{B}$,$\mathcal{C}, \mathcal{D}, \mathcal{E}$, *such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{os\text{-}entauth}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{eufcma}}_{(\mathsf{sig},\mathsf{ver})}(\mathcal{B}) + n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}^{\mathsf{Gap\text{-}DH}}_{E(\mathbb{F}_p)}(\mathcal{C}) \\ + n_S \cdot n_C \cdot \mathbf{Adv}^{\mathsf{int\text{-}0}}_{\mathsf{AE}}(\mathcal{D}) + n_C^2 \cdot \mathbf{Adv}^{\mathsf{CDH}}_{E(\mathbb{F}_p)}(\mathcal{E}),\end{aligned}$$

  where $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ *each run in time* $t + O(\mu)$ *where* μ *is total number of bits queried.*
- *If there exists an adversary* $\mathcal{A}$ *running in time at most* t *against the message authentication property of OS-EAMAP security then there are adversaries* $\mathcal{B}$*,* $\mathcal{C}$ *and* $\mathcal{D}$*, such that*

$$\mathbf{Adv}^{\mathsf{os\text{-}auth}}_{\Pi}(\mathcal{A}) \leq n_S \cdot (n_C + n_T) \cdot \mathbf{Adv}^{\mathsf{intsfptxt}}_{\mathsf{AE}}(\mathcal{D}) + n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}^{\mathsf{Gap\text{-}DH}}_{E(\mathbb{F}_p)}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{os\text{-}entauth}}_{\Pi}(\mathcal{B}),$$

  where $\mathcal{B}$ *runs in time* t *and,* $\mathcal{C}$ *and* $\mathcal{D}$ *each run in time* $t + O(\mu)$ *where* μ *is total number of bits queried.*
- *If there exists an adversary* $\mathcal{A}$ *against the message privacy property of OS-EAMAP security then there are adversaries* $\mathcal{B}$*,* $\mathcal{C}$ *and* $\mathcal{D}$*, such that*

$$\mathbf{Adv}^{\mathsf{os\text{-}priv}}_{\Pi}(\mathcal{A}) \leq n_S \cdot (n_C + n_T) \cdot \mathbf{Adv}^{\mathsf{indsfcca}}_{\mathsf{AE}}(\mathcal{D}) + n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}^{\mathsf{Gap\text{-}DH}}_{E(\mathbb{F}_p),}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{os\text{-}entauth}}_{\Pi}(\mathcal{B}),$$

  where $\mathcal{B}$ *runs in time* t *and,* $\mathcal{C}$ *and* $\mathcal{D}$ *each run in time* $t + O(\mu)$ *where* μ *is total number of bits queried.*

where n_C *is the number of cards in the system,* n_T *the number of terminals,* n_S *the number of sessions and* $|h|$ *is the output size of the hash function.*

Note that int-0 defines security for an adversary against intsfptxt (or intsfctxt) that is permitted no encryption oracle queries.

We now prove each of the entity authentication, message authentication and message privacy properties in turn.

Before proceeding with the main proof we first examine a related concept of Key Secrecy for a simpler protocol, π, described in Figure 5. To analyse this protocol we are only interested in whether the secret key remains secret, and so we introduce a new security game to model this fact. The security definition for Weak Key Secrecy can in found in Appendix D.

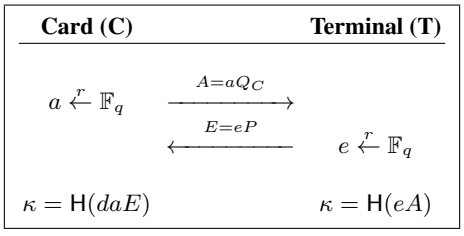

Fig. 5. Unauthenticated Key-Agreement Scheme

Given this definition we can now analyse the protocol in Figure 5. The proof relies on the following problem being hard.

Definition 17. *(Gap Diffie–Hellman) Let $\mathcal{O}_{\mathsf{DDH}}$ be an oracle that solves the DDH problem in G, i.e. takes as input $rP, sP, tP \in G$, and outputs one if $tP = rsP$ and zero otherwise.*
The Gap Diffie–Hellman problem then asks that given $aP, bP \in G$ where $a, b \xleftarrow{r} \mathbb{F}_q$, and access to $\mathcal{O}_{\mathsf{DDH}}$, compute abP (i.e. solve CDH). The advantage of an adversary $\mathcal{A}$ against the Gap Diffie–Hellman problem is defined by

$$\mathbf{Adv}_G^{\mathsf{Gap\text{-}DH}}(\mathcal{A}) = \Pr[a, b \xleftarrow{r} \mathbb{F}_q : \mathcal{A}^{\mathcal{O}_{\mathsf{DDH}}}(aP, bP) = abP].$$

Lemma 1. *The weak key secrecy of the reduced protocol π is reducible to the Gap Diffie–Hellman assumption, i.e. we have for all adversaries $\mathcal{A}$ there exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}_\pi^{\mathsf{wKSec}}(\mathcal{A}) \leq n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}_{E(\mathbb{F}_p)}^{\mathsf{Gap\text{-}DH}}(\mathcal{B}),$$

where n_C is the number of cards in the system and $|h|$ is the output length of the hash function.

Proof. The proof of this lemma uses the technique first presented in [13] for analysing a hashed Diffie–Hellman based key agreement protocol. Assume we have an adversary $\mathcal{A}$ against the key secrecy of π we shall use this to construct an adversary $\mathcal{B}$ against Gap Diffie–Hellman, where $\mathcal{B}$ is given the challenge aP, bP.

The algorithm $\mathcal{B}$ begins by setting up n_C authenticated participants by choosing a secret key $d_i \xleftarrow{r} \mathbb{F}_q$ for each authenticated participant $i \in C$ and sets the public key $Q_i = d_i P$ except for one participant $i^* \in C$ where we set the public key to aP. $\mathcal{B}$ also sets up n_T unauthenticated participants.

Algorithm $\mathcal{B}$ will then use its DDH oracle $\mathcal{O}_{DDH}$ to provide simulations of $\mathcal{A}$'s oracles as follows:

- $\mathsf{NewSession}(i, \rho)$ – $\mathcal{B}$ starts a new session for i. All participants may have a total of n_s sessions.
- $\mathsf{Send}(\pi_i^s, m)$ –
  - For $i \in C$ (and $\rho = \texttt{initiator}$), select at random $\alpha_i^s \xleftarrow{r} \mathbb{F}_q$ to create message $A = \alpha_i^s Q_i$.
  - For $i \in T$ ($\rho = \texttt{responder}$), select at random $\beta_i^s \xleftarrow{r} \mathbb{F}_q$ to create message $E = \beta_i^s bP$.

  This will result in a shared key $\kappa = \mathsf{H}(\alpha_{i*}^s \beta_{i*}^s abP)$ for oracle π_{i*}^s with partner π_j^t, where $j \in T$.
- $\mathsf{Corrupt}(i, d')$ –
  - For $i \in C$, then return d_i and replace it with d' unless $i = i^*$ in which case abort
  - For $i \in T$, return $\perp$.
- $\mathsf{Reveal}(\pi_i^s)$ – To answer Reveal queries, $\mathcal{B}$ will maintain a Guess session key list (G-List). Each element on the G-List is a tuple of the form (τ, i, j, κ_R). Queries are answered as follows:
  - First $\mathcal{B}$ checks the G-list and if there is an entry for i, j then $\mathcal{B}$ outputs the corresponding κ_R.
  - If not then $\mathcal{B}$ checks whether the H-list (see below) contains an (M, h, st_h) with $\mathcal{O}_{DDH}(\alpha_i^s Q_i, \beta_i^s bP, M) = 1$. If it does then $\mathcal{B}$ sets $st_h = \{i, j\}$ and adds to G-list (τ, i, j, h).
  - Otherwise $\mathcal{B}$ returns a randomly chosen key.
- $\mathsf{H}(M)$ – To answer hash queries, $\mathcal{B}$ maintains an H-List containing tuples of the form (M, h, st_h). Queries are answered as followed:
  - $\mathcal{B}$ first checks whether M is on the H-list. If it is, then $\mathcal{B}$ outputs h.
  - If not then $\mathcal{B}$ must check whether $\mathsf{H}(M)$ is already an valid entry on the G-list for some pair of participants (i, j) by calling its $\mathcal{O}_{DDH}$.
  - If it is a valid entry for some pair of participants (i, j) then $\mathcal{B}$ returns the corresponding κ_R from the G-list and adds $(M, \kappa_R, \{i, j\})$ to the H-list.
  - Otherwise $\mathcal{B}$ chooses a random hash h and adds (M, h, st_h) to list.

Eventually, $\mathcal{A}$ will output its guess $\pi^* = \pi_i^s$ and κ^*, The probability that $\mathcal{A}$ chooses $i = i^*$ is $1/n_C$ Note that in this case i^* will not have been corrupted so the simulation has been perfect. At this point $\mathcal{B}$ searches the H-list for the entry $(M^*, \kappa^*, st_\kappa^*)$ corresponding to κ^*, using $\mathcal{O}_{DDH}$ to verify that the entry corresponds to i^*, j. If this entry does not exist then $\mathcal{A}$ must have output a random guess for the key, in which case his probability of success is at best $1/|h|$, where $|h|$ is the size of the output to the function H. Since we assume $\mathcal{A}$ to be a winning adversary with probability $(1 - 1/|h|)$ $\mathcal{A}$ queries H such that his guess is on the H-list. If it is on the list then $\mathcal{B}$ calculates the solution to the gap-DH problem as $(1/\alpha_{i^*}^s \beta_{i^*}^s) M^*$. □

4.1 One-sided Entity Authentication

We now turn to proving the different properties in our main theorem, starting with one-sided entity authentication. We make use of the following definition.

Definition 18. *(Computational Diffie–Hellman) The CDH problem then asks that given $rP, sP \in G$, where $r, s \xleftarrow{r} \mathbb{F}_q$, compute rsP. The advantage of an adversary $\mathcal{A}$ against the CDH problem is defined by*

$$\mathbf{Adv}_G^{\mathsf{CDH}}(\mathcal{A}) = \Pr[r, s \xleftarrow{r} \mathbb{F}_q : \mathcal{A}(rP, sP) = rsP].$$

Lemma 2. *If the gap-DH and CDH problems are hard, $\mathsf{AE} = (\mathsf{enc}, \mathsf{dec})$ is an int-0 secure AE scheme, and the signature scheme $(\mathsf{sig}, \mathsf{ver})$ used to produce card certificates is EUF-CMA, then the EMV protocol $P = \{\Pi, \mathcal{G}\}$ is secure in the sense of OS-EA. In particular, if there exists an adversary $\mathcal{A}$ against $P = \{\Pi, \mathcal{G}\}$ in the sense of* os-entauth *then there are adversaries $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$ such that*

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{A}) \leq\ & \mathbf{Adv}_{(\mathsf{sig},\mathsf{ver})}^{\mathsf{eufcma}}(\mathcal{B}) + n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}_{E(\mathbb{F}_p)}^{\mathsf{Gap\text{-}DH}}(\mathcal{C}) \\ & + n_S \cdot n_C \cdot \mathbf{Adv}_{\mathsf{AE}}^{\mathsf{int\text{-}0}}(\mathcal{D}) + n_C^2 \cdot \mathbf{Adv}_{E(\mathbb{F}_p)}^{\mathsf{CDH}}(\mathcal{E}),\end{aligned}$$

where n_C is the number of cards in the system, n_S the number of sessions and $|h|$ is the output size of the hash function.

Proof. Let $\mathcal{A}$ be an adversary that wins the os-entauth experiment, that is the adversary succeeds in getting one of the bad events os-notval, os-notconf, notuni to occur. Then using the inequality $\Pr[A \vee B] \leq \Pr[A] + \Pr[B|\neg A]$, we obtain:

$$\begin{aligned}\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{A}) &= \Pr[\mathsf{os\text{-}notval} \vee \mathsf{os\text{-}notconf} \vee \mathsf{notuni}] \\ &\leq \Pr[\mathsf{os\text{-}notconf}] + \Pr[\mathsf{os\text{-}notval} \vee \mathsf{notuni} | \neg \mathsf{os\text{-}notconf}] \\ &\leq \Pr[\mathsf{os\text{-}notconf}] + \Pr[\mathsf{notuni}] + \Pr[\mathsf{os\text{-}notval} | \neg \mathsf{os\text{-}notconf} \wedge \neg \mathsf{notuni}].\end{aligned}$$

Let us analyse these terms in turn:

(i) $\Pr[\mathsf{os\text{-}notconf}]$:
To win in this case an adversary must successfully impersonate a valid card, i.e. there exists $\Pi_{i^*}^s$ ($i^* \in T$) which accepts with $pid = j^*$ and session identifier sid^*, for which there exists no oracle $\Pi_{j^*}^t$ with $pid = i^*$ and session identifier sid^*. More specifically $\mathcal{A}$ must send a valid key confirmation message $\mathsf{enc}_{\kappa^*}(cert_{Q^*}, a^*, Q^*; st_e)$. Let F be event that $cert_{Q^*}$ is a valid certificate forgery, i.e. it was not output by the the sig algorithm during the setup phase.

$$\Pr[\mathsf{os\text{-}notconf}] \leq \Pr[\mathsf{os\text{-}notconf} \wedge F] + \Pr[\mathsf{os\text{-}notconf} \wedge \neg F]$$

Consider each term in turn:

$\Pr[\mathsf{os\text{-}notconf} \wedge F]$:
We shall use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against the EUF-CMA property of the signature scheme that the card issuer used to sign the certificate.
Algorithm $\mathcal{B}$ begins by setting up n_C authenticated participants by choosing secret keys $d_i \in \mathbb{F}_q$ and sets the public key to be $Q_i = d_i P$. Additionally $\mathcal{B}$ calls his sign oracle to generate the certificates for these public keys. $\mathcal{B}$ also sets up n_T unauthenticated participants. $\mathcal{B}$ models $\mathcal{A}$'s NewSession, Reveal, Corrupt, Send queries appropriately using the key material he has generated.
At some point $\mathcal{A}$ issues a query $\mathsf{Send}(\Pi_{j^*}^t, a^* d^* P, \mathtt{type})$, where $d^* P$ is a "forged" public key for some $j^* \in C$ such that a^* and d^* were chosen by $\mathcal{A}$. $\mathcal{B}$ shall responds by generating the ephemeral public key for terminal i^*, specifically $e^* P$ for some $e^* \in \mathbb{F}_q$. Now $\mathcal{A}$ and the simulated terminal oracle $\Pi_{i^*}^s$

have derived a key $\kappa^* = \mathsf{H}(a^*d^*e^*P)$. Next in order for $\mathcal{A}$ to get Π^s_{i*} to accept he must responding with $\mathsf{enc}_{\kappa^*}(\mathrm{cert}_{j^*}, a^*, d^*P)$ such that cert_{j^*} verifies correctly. Upon receipt of $\mathsf{enc}_{\kappa^*}(\mathrm{cert}_{j^*}, a^*, d^*P)$, $\mathcal{B}$ decrypts using κ^* and then outputs $(d^*P, \mathrm{cert}_{j^*})$ as his forgery. Therefore,

$$\Pr[\mathsf{os\text{-}notconf} \wedge F] \leq \mathbf{Adv}^{\mathsf{eufcma}}_{\mathrm{cert}}(\mathcal{B}).$$

$\Pr[\mathsf{os\text{-}notconf} \wedge \neg F]$:
Let H be the event that $\mathcal{A}$ makes a hash query which reveals the session key.

$$\Pr[\mathsf{os\text{-}notconf} \wedge \neg F] \leq \Pr[\mathsf{os\text{-}notconf} \wedge \neg F \wedge H] + \Pr[\mathsf{os\text{-}notconf} \wedge \neg F \wedge \neg H]$$

First consider $\Pr[\mathsf{os\text{-}notconf} \wedge \neg F \wedge H]$.
We shall use $\mathcal{A}$ to construct an adversary $\mathcal{C}'$ against the wKSec property of the unauthenticated key exchange protocol π (cf. Figure 5). Adversary $\mathcal{C}'$ simulates the environment for $\mathcal{A}$ and begins by calling it's setup algorithm to initialise n_C authenticated participants and n_T unauthenticated participants. $\mathcal{C}'$ models $\mathcal{A}$'s $\mathsf{NewSession}_{\mathcal{A}}$, $\mathsf{Reveal}_{\mathcal{A}}$, $\mathsf{Corrupt}_{\mathcal{A}}$, $\mathsf{Send}_{\mathcal{A}}$ queries appropriately by making the corresponding queries to it's own challenger, i.e. $\mathsf{NewSession}_{\mathcal{C}'}$, $\mathsf{Reveal}_{\mathcal{C}'}$, $\mathsf{Corrupt}_{\mathcal{C}'}$, $\mathsf{Send}_{\mathcal{C}'}$. If $\mathcal{A}$ makes a $\mathsf{Send}_{\mathcal{A}}(\Pi^s_i, m, \mathtt{type})$ query where $\mathtt{type} = \mathtt{app}$ or $\mathtt{ch}$ then $\mathcal{C}'$ will first make a $\mathsf{Reveal}_{\mathcal{C}'}$ query and then performs the necessary encryption or decryption itself. Note that the key revealed will only be forwarded back to $\mathcal{A}$ if he issues the same query to $\mathsf{Reveal}_{\mathcal{A}}$. If $\mathcal{A}$ makes a hash query $\mathsf{H}(m)$, $\mathcal{C}'$ will forward this query to his hash oracle but maintains an H-list of each message and hash, (m, h).
In order to win $\mathcal{A}$ must deduce the session key it has established with Π^s_{i*} prior to performing any encryption operations, i.e. before the key confirmation step. Therefore, $\mathcal{C}'$ will not have issued a reveal query to Π^s_{i*} and so the key that $\mathcal{A}$ determines will not violate any of $\mathcal{C}'$'s winning conditions. At some point $\mathcal{A}$ achieves its goal and Π^t_{j*} accepts. Thus, $\mathcal{A}$ has made a query to H which revealed the session key for Π^t_{j*}. $\mathcal{C}'$ can therefore check which h on his H-list decrypts the confirmation message $\mathcal{A}$ sent to Π^t_{j*} correctly. $\mathcal{C}'$ outputs $\kappa^* = h$ and Π^t_{j*}. Therefore (by Lemma 1),

$$\Pr[\mathsf{os\text{-}notconf} \wedge \neg F \wedge H] \leq \mathbf{Adv}^{\mathsf{wKSec}}_{\pi}(\mathcal{C}') \leq (1 - 1/|h|) \cdot n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}^{\mathsf{Gap\text{-}DH}}_{E(\mathbb{F}_p)}(\mathcal{C}).$$

Next consider $\Pr[\mathsf{os\text{-}notconf} \wedge \neg F \wedge \neg H]$.
We shall use $\mathcal{A}$ to construct a new adversary $\mathcal{D}$ against the INT-0 security of AE, (where INT-0 is the normal INT-sfPTXT game but the adversary is permitted no encryption queries). $\mathcal{D}$ begins by guessing for which session s^*, card $i^* \in C$ he thinks $\mathcal{A}$ will impersonate $i^* \in C$ successfully. What we effectively do is set the output of the random oracle H for the key corresponding to $\Pi^{s^*}_{i^*}$ (and its partner $\Pi^{t^*}_{j^*}$ respectively) to be the key chosen at random for the INT-0 experiment. All other keys are initialised by $\mathcal{D}$ appropriately. If $\mathcal{A}$ makes a Send query with $\mathtt{type} = \mathtt{app}$ after $\delta = \mathtt{accept}$ for $\Pi^{s^*}_{i^*}$ then $\mathcal{D}$ shall abort since he is not permitted any encryption queries (similarly it is up to $\mathcal{A}$ to make an appropriate ciphertext forgery for the encrypted confirmation message). All other queries $\mathsf{NewSession}$, Reveal, $\mathsf{Corrupt}$ and Send for $\Pi^s_i \neq \Pi^{s^*}_{i^*}$ are simulated internally by $\mathcal{D}$ using appropriate randomness. Since $\mathcal{A}$ does not make any reveal queries or hash queries corresponding to the key of $\Pi^{s^*}_{i^*}$ the simulation shall remain perfect. When $\mathcal{A}$ outputs a valid key confirmation message then $\mathcal{D}$ has a valid ciphertext forgery.

$$\Pr[\mathsf{os\text{-}notconf} \wedge \neg F \wedge \neg H] \leq n_S \cdot n_C \cdot \mathbf{Adv}^{\mathsf{int\text{-}0}}_{\mathsf{AE}}(\mathcal{D}).$$

(ii) $\Pr[\mathsf{notuni}]$:
Here we must consider the case when two card sessions establish the same key with a single terminal.
Let $\mathcal{A}$ be an adversary against the uniqueness of sessions. We shall use $\mathcal{A}$ to construct a new adversary $\mathcal{E}$ that solves the CDH problem given challenge rP, sP.
Algorithm $\mathcal{E}$ begins by setting up n_C authenticated participants by choosing secret keys $d_i \in \mathbb{F}_q$ for each authenticated participant and sets the public keys to be $Q_i = d_iP$. Except for two cards C_1 and C_2 chosen

at random (note we also consider the case that $C_1 = C_2$). First $\mathcal{E}$ chooses d, a_1 and a_2 at random from $\mathbb{F}_q$. Next $\mathcal{E}$ sets the public key of C_1 to be $Q_1 = a_1^{-1}drP$ and (when $C_1 \neq C_2$) the public key of C_2 to be $Q_2 = a_2^{-1}dP$.
$\mathcal{E}$ models $\mathcal{A}$'s NewSession, Reveal, Corrupt, Send queries appropriately using the key material it has generated and necessary randomness. Except for cards C_1 and C_2 where Send queries are modelled such that:

- C_1 first sends $a_1 Q_1 = drP$ to some terminal T_j.
- T_j responds with sP
- The key established between C_1 and T_j is $\mathsf{H}(d(rsP))$. (To model any further send queries with $\texttt{type} = \texttt{app}$ or $\texttt{ch}$ $\mathcal{E}$, chooses this hash uniformly at random and uses this as the key to perform the necessary encryptions and decryptions.)
- Next start a new session for C_2 by sending dP. In the case of $C_1 \neq C_2$ this corresponds to $a_2 Q_2$ and in the case of $C_1 = C_2$ this corresponds to $a_2' Q_1$ for some $a_2' \xleftarrow{r} \mathbb{F}_q$.

Finally, adversary $\mathcal{A}$ must impersonate the terminal and send rsP to C_2. This will ensure that C_2 establishes the same session key as the previous session of C_1 and T_j, ($\kappa = \mathsf{H}(drsP)$). The adversary $\mathcal{E}$ then uses $\mathcal{A}$'s impersonated terminal message rsP as its CDH solution.

$$\Pr[\mathsf{notuni}] \leq n_C^2 \cdot \mathbf{Adv}^{\mathsf{CDH}}_{E(\mathbb{F}_p)}(\mathcal{E}).$$

(iii) $\Pr[\mathsf{os\text{-}notval} | \neg \mathsf{os\text{-}notconf} \wedge \neg \mathsf{notuni}]$: If neither of the bad events os-notconf or notuni occur then all "completed" oracles have accepted and they have unique partners. By the construction of Π (cf. Figure 4) this means that the key confirmation has been performed correctly and hence each oracle shares the same key with their corresponding partner. Therefore the event os-notval will not occur.

$$\Pr[\mathsf{os\text{-}notval} | \neg \mathsf{os\text{-}notconf} \wedge \neg \mathsf{notuni}] = 0.$$

□

4.2 One-sided Message Authentication

We now turn to the message authentication property:

Lemma 3. *If the Gap-DH problem is hard in $E(\mathbb{F}_p)$, $\mathsf{AE} = (\mathsf{enc}, \mathsf{dec})$ is an int-sfptxt secure AE scheme and $P = \{\Pi, \mathcal{G}\}$ is secure in the sense of* os-entauth*, then $P = \{\Pi, \mathcal{G}\}$ is secure in the sense of* os-auth*. In particular if there is an adversary $\mathcal{A}$ against the* os-auth *property then there are adversaries $\mathcal{B}, \mathcal{C}$ and $\mathcal{D}$ such that*

$$\mathbf{Adv}^{\mathsf{os\text{-}auth}}_{\Pi}(\mathcal{A}) \leq n_S \cdot (n_C + n_T) \cdot \mathbf{Adv}^{\mathsf{intsfptxt}}_{\mathsf{AE}}(\mathcal{D}) + n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}^{\mathsf{Gap\text{-}DH}}_{E(\mathbb{F}_p)}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{os\text{-}entauth}}_{\Pi}(\mathcal{B}),$$

where n_C is the number of cards in the system, n_T the number of terminals, n_S the number of sessions and $|h|$ is the output size of the hash function.

Proof. We shall prove this result via a sequence of games. Let $\mathcal{A}$ be adversary attacking Π in the sense of auth.
Game 0: This game is identical to $\mathbf{Exec}^{\mathsf{os\text{-}auth}}_{\Pi}(\mathcal{A})$.

$$\Pr[\mathsf{Game0} \Rightarrow 1] = \mathbf{Adv}^{\mathsf{os\text{-}auth}}_{\Pi}(\mathcal{A}).$$

Game 1: This proceeds identically to the previous game but aborts if a terminal ($i \in T$) oracle Π_i^s accepts but has no unique partner oracle which establishes the same key. It is easy to see that if this event occurs then one of the events os-notval, os-notconf, notuni has occurred. Therefore,

$$\Pr[\mathsf{Game0} \Rightarrow 1] \leq \Pr[\mathsf{Game1} \Rightarrow 1] + \mathbf{Adv}^{\mathsf{os\text{-}entauth}}_{\Pi}(\mathcal{B}).$$

Game 2: This proceeds identically to the previous game but aborts if $\mathcal{A}$ makes a query to H which reveals the key for an oracle Π_i^s. Again it is easy to see that

$$\Pr[\mathsf{Game1} \Rightarrow 1] \leq \Pr[\mathsf{Game2} \Rightarrow 1] + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{C}').$$

Game 3: The challenger now selects at random an oracle $\Pi_{i^*}^{s^*}$. The game aborts if $\mathsf{Prefix}(L_{i,s}^{app|rec}, L_{j,t}^{app|sen}) = \mathsf{false}$ for $(i,s) \neq (i^*, s^*)$. Since i^* is chosen at random from $I = C \cup T$ we have:

$$\Pr[\mathsf{Game2} \Rightarrow 1] \leq n_S \cdot (n_C + n_T) \cdot \Pr[\mathsf{Game3} \Rightarrow 1].$$

It remains to study the probability that $\mathcal{A}$ wins ($\mathsf{Game3} \Rightarrow 1$). We shall use $\mathcal{A}$ in Game3 to construct a new adversary $\mathcal{D}$ against the INT-sfPTXT security of AE. What we effectively do is set the output of the random oracle H for the key corresponding to $\Pi_{i^*}^{s^*}$ to be the key chosen at random for the INT-sfPTXT experiment. The adversary $\mathcal{D}$ will use its enc and dec oracles to provide simulations of $\mathcal{A}$'s Send queries with $\Pi_{i^*}^{s^*}$ or $\Pi_{j^*}^{t^*}$ (where $\mathsf{P}(\Pi_{i^*}^{s^*}, \Pi_{j^*}^{t^*}) = \mathsf{true}$) as the input session. All other queries NewSession, Reveal, Corrupt and all other Send queries will be simulated internally by $\mathcal{D}$. We further explain how Send queries for $\Pi_{i^*}^{s^*}$ or $\Pi_{j^*}^{t^*}$ are simulated. When $\mathcal{A}$ makes a Send query for $\Pi_{i^*}^{s^*}$ or $\Pi_{j^*}^{t^*}$ before $\delta = \mathtt{derived}$, $\mathcal{D}$ performs the key exchange himself by using appropriate randomness. Once $\delta = \mathtt{derived}$, $\mathcal{D}$ must make calls to enc and dec in order to perform the key confirmation. Since the confirmation message received/sent by $\Pi_{i^*}^{s^*}$ should be equal to that sent/received by the partnered card/terminal session respectively, when $\mathcal{D}$ calls dec he will not be given the plaintext due to restrictions on decryption queries in the INT-sfPTXT model. If this happens then he accepts the card session and sets $\delta = \mathtt{accept}$, otherwise he sets $\delta = \mathtt{reject}$. After $\delta = \mathtt{accept}$ whenever $\mathcal{A}$ makes a Send query where $\mathtt{type} = \mathtt{app}$ or $\mathtt{ch}$, $\mathcal{D}$ calls his enc or dec oracle, respectively. Since $\mathcal{A}$ does not make any reveal queries or hash queries corresponding to the key of $\Pi_{i^*}^{s^*}$ the simulation remains perfect. If $\mathcal{A}$ wins the auth game then $\mathsf{Prefix}(L_{i^*,s^*}^{app|rec}, L_{j^*,t^*}^{app|sen}) = \mathsf{false}$ and thus $\mathcal{A}$ has output a ciphertext forgery which allows $\mathcal{D}$ to win the INT-sfPTXT game.

$$\Pr[\mathsf{Game3} \Rightarrow 1] \leq \mathbf{Adv}_{\mathsf{AE}}^{\mathsf{intsfptxt}}(\mathcal{D}).$$

Combining all of the above we obtain

$$\begin{aligned}
\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}auth}}(\mathcal{A}) &= \Pr[\mathsf{Game0} \Rightarrow 1] \\
&\leq \Pr[\mathsf{Game1} \Rightarrow 1] + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B}) \\
&\leq \Pr[\mathsf{Game2} \Rightarrow 1] + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{C}') + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B}) \\
&\leq n_S \cdot (n_C + n_T) \cdot \Pr[\mathsf{Game3} \Rightarrow 1] + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{C}') + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B}) \\
&\leq n_S \cdot (n_C + n_T) \cdot \mathbf{Adv}_{\mathsf{AE}}^{\mathsf{intsfptxt}}(\mathcal{D}) + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{C}') + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B})
\end{aligned}$$

With the final result following from applying Lemmas 1 and 2.

4.3 One-sided Message Privacy

We now turn to the message privacy property:

Lemma 4. *If the Gap-DH problem is hard in $E(\mathbb{F}_p)$, $\mathsf{AE} = (\mathsf{enc}, \mathsf{dec})$ is an ind-sfcca secure AE scheme and $P = \{\Pi, \mathcal{G}\}$ is secure in the sense of* os-entauth. *Then $P = \{\Pi, \mathcal{G}\}$ is secure in the sense of* os-priv*, i.e. any adversary $\mathcal{A}$ against the* os-priv *property can be turned into adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ such that*

$$\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}priv}}(\mathcal{A}) \leq n_S \cdot (n_C + n_T) \cdot \mathbf{Adv}_{\mathsf{AE}}^{\mathsf{indsfcca}}(\mathcal{D}) + n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}_{E(\mathbb{F}_p),}^{\mathsf{Gap\text{-}DH}}(\mathcal{C}) + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B}),$$

Proof. We shall prove this result via a sequence of games. Let $\mathcal{A}$ be adversary attacking Π in the sense of priv.
Game 0: This game is identical to $\mathbf{Exec}_{\Pi}^{\mathsf{os\text{-}priv}}(\mathcal{A})$.

$$\Pr[\mathsf{Game0} \Rightarrow 1] - \frac{1}{2} = \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}priv}}(\mathcal{A}).$$

Game 1: This proceeds identically to the previous game but aborts if a terminal ($i \in T$) oracle Π_i^s accepts but has no partner oracle that establishes the same key. It is easy to see that if this event occurs then one of the bad events $\mathsf{os\text{-}notval}, \mathsf{os\text{-}notconf}, \mathsf{notuni}$ has occurred. Therefore,

$$\Pr[\mathsf{Game0} \Rightarrow 1] \leq \Pr[\mathsf{Game1} \Rightarrow 1] + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B}).$$

Game 2: This proceeds identically to the previous game but aborts if $\mathcal{A}$ makes a query to H which reveals the key for an oracle Π_i^s. Again it is easy to see that

$$\Pr[\mathsf{Game1} \Rightarrow 1] \leq \Pr[\mathsf{Game2} \Rightarrow 1] + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{C}').$$

Game 3: The challenger now selects at random an oracle $\Pi_{i^*}^{s^*}$. The game aborts if the attacker outputs (i, s, b') such that $(i, s) \neq (i^*, s^*)$, the game will instead return a random bit. Since i^* is chosen at random from $I = C \cup T$ we have:

$$\Pr[\mathsf{Game2} \Rightarrow 1] - \frac{1}{2} \leq n_S \cdot (n_C + n_T) \cdot \left(\Pr[\mathsf{Game3} \Rightarrow 1] - \frac{1}{2} \right).$$

It remains to study the probability that $\mathcal{A}$ wins ($\mathsf{Game3} \Rightarrow 1$). We shall use $\mathcal{A}$ in $\mathsf{Game3}$ to construct a new adversary $\mathcal{D}$ against the IND-sfCCA security of AE. What we effectively do is set the output of the random oracle H for the key corresponding to $\Pi_{i^*}^{s^*}$ to be the key chosen at random for the IND-sfCCA experiment. The adversary $\mathcal{D}$ will use its enc and dec oracles to provide simulations of $\mathcal{A}$'s Send queries with $\Pi_{i^*}^{s^*}$ or $\Pi_{j^*}^{t^*}$ (where $\mathsf{P}(\Pi_{i^*}^{s^*}, \Pi_{j^*}^{t^*}) = \mathsf{true}$) as the input session. All other queries $\mathsf{NewSession}, \mathsf{Reveal}, \mathsf{Corrupt}$ and all other Send queries will be simulated internally by $\mathcal{D}$. We further explain how Send queries for $\Pi_{i^*}^{s^*}$ or $\Pi_{j^*}^{t^*}$ are simulated. When $\mathcal{A}$ makes a Send query for $\Pi_{i^*}^{s^*}$ or $\Pi_{j^*}^{t^*}$ before $\delta = \texttt{derived}$, $\mathcal{D}$ performs the key exchange himself by using appropriate randomness. Once $\delta = \texttt{derived}$, $\mathcal{D}$ must make calls to enc and dec in order to perform the key confirmation (note $\mathcal{D}$ will give enc the same message in both of its inputs). Since the confirmation message received/sent by $\Pi_{i^*}^{s^*}$ should be equal to that sent/received by the partnered card/terminal session respectively, when $\mathcal{D}$ calls dec he will not be given the plaintext due to restrictions on decryption queries in the IND-sfCCA model. If this happens then he accepts the card session and sets $\delta = \texttt{accept}$, otherwise he sets $\delta = \texttt{reject}$. After $\delta = \texttt{accept}$ whenever $\mathcal{A}$ makes a Send query where $\texttt{type} = \texttt{app}$ or $\texttt{ch}$, $\mathcal{D}$ shall call his enc or dec oracle, respectively. Since $\mathcal{A}$ does not make any reveal queries or hash queries corresponding to the key of $\Pi_{i^*}^{s^*}$ the simulation shall remain perfect. When $\mathcal{A}$ outputs its guess (i^*, s^*, b'), $\mathcal{D}$ shall forward b' as its guess. We therefore have,

$$\Pr[\mathsf{Game3} \Rightarrow 1] - \frac{1}{2} \leq \mathbf{Adv}_{\mathsf{AE}}^{\mathsf{indsfcca}}(\mathcal{D}).$$

Combining all of the above, we yield:

$$\begin{aligned}
\mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}priv}}(\mathcal{A}) &= \Pr[\mathsf{Game0} \Rightarrow 1] - \frac{1}{2} \\
&\leq \Pr[\mathsf{Game1} \Rightarrow 1] - \frac{1}{2} + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B}) \\
&\leq \Pr[\mathsf{Game2} \Rightarrow 1] - \frac{1}{2} + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{C}') + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B}) \\
&\leq n_S \cdot (n_C + n_T) \cdot \left(\Pr[\mathsf{Game3} \Rightarrow 1] - \frac{1}{2} \right) + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{C}') + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B}) \\
&\leq n_S \cdot (n_C + n_T) \cdot \mathbf{Adv}_{\mathsf{AE}}^{\mathsf{indsfcca}}(\mathcal{D}) + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{C}') + \mathbf{Adv}_{\Pi}^{\mathsf{os\text{-}entauth}}(\mathcal{B})
\end{aligned}$$

Again the final result follows from applying Lemmas 1 and 2.

4.4 Unlinkability

Finally, we present our theorem for unlinkability:

Theorem 2. *If gap-DH is hard and* $\mathsf{AE} = (\mathsf{enc}, \mathsf{dec})$ *is an ind-sfcca secure AE scheme, then* $P = \{\Pi, \mathcal{G}\}$ *is secure in the sense of* unlink*; in particular we have*

$$\mathbf{Adv}^{\mathsf{unlink}}_{\Pi}(\mathcal{A}) \leq n_C^2 \cdot \left(\mathbf{Adv}^{\mathsf{indsfcca}}_{\mathsf{AE}}(\mathcal{C}) + n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}^{\mathsf{Gap\text{-}DH}}_{E(\mathbb{F}_p)}(\mathcal{B})\right)$$

where, again, n_C *is the number of cards in the system and* $|h|$ *is the output size of the hash function.*

Note that if a were instead chosen to be of size 2^{32} (as suggested by the RFC) our security analysis would show only 16-bits of security. We refer the reader to the proof in Appendix E for further details.

5 Acknowledgements

This work was support in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO. The second author was also partially supported by a Royal Society Wolfson Merit Award. Research supported in part by the Israel Ministry of Science and Technology (grant 3-9094) and by the Israel Science Foundation (grant 1155/11 and grant 1076/11).

References

1. Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm. *ACM Trans. Inf. Syst. Secur.*, 7(2):206–241, 2004.
2. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
3. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 2000.
4. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.
5. Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In Michael Darnell, editor, *IMA Int. Conf.*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 1997.
6. Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 682–699. Springer, 2012.
7. Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: Relaxed yet composable security notions for key exchange. *IACR Cryptology ePrint Archive*, 2012:242, 2012.
8. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.
9. Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, 2012. Online first; print version to appear.
10. Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J.Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium – 2012*, pages 205–220, 2012.
11. Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. *IACR Cryptology ePrint Archive*, 2011:219, 2011.
12. 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, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, 2012.
13. Caroline Kudla and Kenneth G. Paterson. Modular security proofs for key agreement protocols. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 549–565. Springer, 2005.
14. EMVCo LLC. EMV deployment statistics. *http://www.emvco.com/about_emvco.aspx?id=202*, 2012.
15. EMVCo LLC. EMV ECC key establishment protocols. *http://www.emvco.com/specifications.aspx?id=243*, 2012.
16. Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. The TLS handshake protocol: A modular analysis. *J. Cryptology*, 23(2):187–223, 2010.

17. 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, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389. Springer, 2011.
18. John Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.
19. Phillip Rogaway and Till Stegers. Authentication without elision: Partially specified protocols, associated data, and cryptographic models described by code. In *CSF*, pages 26–39. IEEE Computer Society, 2009.

A Basic Security Definitions

The underlying AE scheme we assume satisfies the following two properties which are variants of the stateful security models of Bellare et al. [1] and Paterson et al. [17].

An adversary against a stateful encryption scheme needs to be given the capability to progress the scheme's state without trivially winning the security experiment. It is for this reason that there is a subtle difference between the standard notions of Indistinguishability under Chosen-Ciphertext Attack (IND-CCA) and Integrity of Plaintexts (INT-PTXT)[2], and their stateful versions IND-sfCCA and INT-sfPTXT. An adversary against IND-sfCCA and INT-sfPTXT security is permitted to query the decryption oracle with an output from the encryption oracle in order to progress the state but the output of this query should *not* be returned to the adversary (to avoid the trivial attack).

In order to match with our security definitions of auth and priv we alter previous IND-sfCCA of Bellare et al. [1] and introduce a new notion for INT-sfPTXT (which is analogous to the definition of INT-sfCTXT given by Bellare at al.) to now compare the lists L^e and L^d. In the case of IND-sfCCA, L^e is the list of all ciphertexts output by the encryption oracle and L^d is the list of all ciphertexts *successfully* decrypted by the decryption oracle. For INT-sfPTXT the lists L^e, L^d correspond to plaintext messages input to and output from the encryption and decryption oracle, respectively. In order to prevent an adversary trivially winning he is not permitted to see the output of the decryption oracle if L^d is a prefix of L^e, i.e. if all ciphertexts decrypted so far were output by the encryption oracle in the case of IND-sfCCA or if all plaintexts output so far were inputs to the encryption oracle in the case of INT-sfPTXT.

Definition 19. *(IND-sfCCA) Consider the scheme* $\mathsf{AE} = \{\mathsf{enc}_\kappa, \mathsf{dec}_\kappa\}$. *Let* $\mathcal{A}$ *be an adversary with access to a left-or-right encryption oracle* $\mathsf{enc}_\kappa(h, \mathsf{LR}_b(m_0, m_1); st_e)$ *and a decryption oracle* $\mathsf{dec}_\kappa(h, c; st_d)$. *It is mandated that any two messages queried to* $\mathsf{enc}_\kappa(h, \mathsf{LR}_b(m_0, m_1); st_e)$ *have equal length. We define an experiment as follows:*

$\mathbf{Exec}^{\mathsf{indsfcca\text{-}}b}_{\mathsf{AE}}(\mathcal{A})$
$\kappa \xleftarrow{r} \{0,1\}^k$, $L^e := \emptyset$, $L^d := \emptyset$, $st_e := \emptyset$ *and*
$st_d := \emptyset$
Run $\mathcal{A}^{\mathsf{enc}_\kappa, \mathsf{dec}_\kappa}$
  Reply to $\mathsf{enc}_\kappa(h, \mathsf{LR}_b(m_0, m_1); st_e)$ *as follows:*
    $(c, st_e) \xleftarrow{r} \mathsf{enc}_\kappa(h, m_b; st_e)$
    $L^e \leftarrow L^e \cup c$; $\mathcal{A} \Leftarrow c$
  Reply to $\mathsf{dec}_\kappa(h, c; st_d)$ *as follows:*
    $(m, st_d) \xleftarrow{r} \mathsf{dec}_\kappa(h, c; st_d)$
    if $m \neq \perp$ ***then***
      $L^d \leftarrow L^d \cup c$
      if $\mathsf{Prefix}(L^d, L^e) = \mathsf{false}$ ***then*** $\mathcal{A} \Leftarrow m$
Until $\mathcal{A}$ *returns a bit* b'
return b'

The attacker wins when $b' = b$, *and his advantage is defined as*

$$\mathbf{Adv}^{\mathsf{indsfcca}}_{\mathsf{AE}}(\mathcal{A}) = \Pr[\mathbf{Exec}^{\mathsf{indsfcca\text{-}1}}_{\mathsf{AE}}(\mathcal{A}) = 1] - \Pr[\mathbf{Exec}^{\mathsf{indsfcca\text{-}0}}_{\mathsf{AE}}(\mathcal{A}) = 1].$$

Definition 20. *(INT-sfPXT) Consider the scheme* $\mathsf{AE} = \{\mathsf{enc}_\kappa, \mathsf{dec}_\kappa\}$. *Let* $\mathcal{A}$ *be an adversary with oracle access to* $\mathsf{enc}_\kappa(h, m; st_e)$ *and* $\mathsf{dec}_\kappa(h, c; st_d)$. *We define an experiment as follows:*

$\mathbf{Exec}^{\mathsf{intsfptxt}}_{\mathsf{AE}}(\mathcal{A})$
$\kappa \xleftarrow{r} \{0,1\}^k$, $d := 0$
$L^e := \emptyset$, $L^d := \emptyset$, $st_e := \emptyset$ *and* $st_d := \emptyset$
Run $\mathcal{A}^{\mathsf{enc}_\kappa, \mathsf{dec}_\kappa}$
  Reply to $\mathsf{enc}_\kappa(h, m; st_e)$ *as follows:*
    $(c, st_e) \xleftarrow{r} \mathsf{enc}_\kappa(h, m; st_e)$
    $L^e \leftarrow L^e \cup m$; $\mathcal{A} \Leftarrow c$
  Reply to $\mathsf{dec}_\kappa(h, c; st_d)$ *as follows:*
    $(m, st_d) \xleftarrow{r} \mathsf{dec}_\kappa(h, c; st_d)$
    if $m \neq \perp$ ***then***
      $L^d \leftarrow L^d \cup m$; $\mathcal{A} \Leftarrow 1$
      if $\mathsf{Prefix}(L^d, L^e) = \mathsf{false}$ ***then*** $d := 1$
    else $\mathcal{A} \Leftarrow 0$
Until $\mathcal{A}$ *halts*
return d

The advantage $\mathbf{Adv}^{\mathsf{intsfptxt}}_{\mathsf{AE}}(\mathcal{A})$ *of an adversary is defined as*

$$\mathbf{Adv}^{\mathsf{intsfptxt}}_{\mathsf{AE}}(\mathcal{A}) = \Pr[\mathbf{Exec}^{\mathsf{intsfptxt}}_{\mathsf{AE}}(\mathcal{A}) = 1].$$

In addition we define the related notion int-0 which considers an adversary $\mathcal{A}$ against intsfptxt (or in fact intsfctxt) that is permitted no encryption oracle queries.

Definition 21. *(EUF-CMA) Consider the signature scheme* $\{\mathsf{keysig}, \mathsf{sig}, \mathsf{ver}\}$, *where* keysig *be the key generation method for this scheme. Let* $\mathcal{A}$ *be an adversary that has access to the oracle* $\mathsf{sig}_{sk}(\cdot)$. *We define the experiment as follows:*

$\mathbf{Exec}^{\mathsf{eufcma}}_{(\mathsf{sig},\mathsf{ver})}(\mathcal{A})$
$(pk, sk) \xleftarrow{r} \mathsf{keysig}$
$(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{sig}_{sk}(\cdot)}$
if $\mathsf{ver}_{pk}(m, \sigma) = 1$*; and* m *has not been queried to* $\mathsf{sig}_{sk}(\cdot)$
then return *1* ***else return*** *0*

The attacker's advantage is defined as

$$\mathbf{Adv}^{\mathsf{eufcma}}_{(\mathsf{sig},\mathsf{ver})}(\mathcal{A}) = \Pr[\mathbf{Exec}^{\mathsf{eufcma}}_{(\mathsf{sig},\mathsf{ver})}(\mathcal{A}) = 1].$$

B Jager et al.'s Definition of ACCE

Here we present the revised ACCE definition of Jager et al. [11]. In this definition each oracle Π^s_i maintains an additional internal state variable $b^s_i \xleftarrow{r} \{0,1\}$ chosen at random at the start of the game. Further to this an oracle Π^s_i maintains variables $(u^s_i, v^s_i, c^s_i, \theta^s_i)$. The states u^s_i and v^s_i are counters (initialised to $(0,0)$) used to ensure that $\mathcal{A}$ cannot submit a ciphertext previously output by Encrypt oracle to the Decrypt oracle. The variable c^s_i defines the list of ciphertext output by the encryption oracle, where $c^s_i[u]$ denotes the u-th entry on the list. Finally, θ^s_i stores the pair indices (j,t) necessary to define the partner Π^t_j of Π^s_i. The two states st_e and st_d are maintained by encryption and decryption operations of the stateful symmetric encryption scheme (each oracle Π^s_i shall maintain a different set of states). As before we let enc and dec be the encryption and decryption algorithms of our symmetric encryption scheme. The adversary $\mathcal{A}$ will be permitted to make the following queries:

- $\mathsf{Send}^{\mathrm{pre}}(\Pi_i^s, m)$: This is identical to the Send query in the preliminaries section above, except that it replies with $\perp$ if oracle Π_i^s has state $\delta = \texttt{accept}$ (this shall be handled by the decrypt query).
- $\mathsf{Reveal}(\Pi_i^s)$ and $\mathsf{Corrupt}(i)$ are the standard queries for revealing a session key and corrupting a participant.
- $\mathsf{Encrypt}(\Pi_i^s, m_0, m_1, h)$: takes as input two equal length messages m_0 and m_1 and a header h. If Π_i^s has $\delta \neq \texttt{accept}$ then Π_i^s returns $\perp$. Otherwise it proceeds with encryption as in Figure 6 dependent on the internal state b_i^s.
- $\mathsf{Decrypt}(\Pi_i^s, c, h)$: takes as input a ciphertext c and a header h. If Π_i^s has $\delta \neq \texttt{accept}$ then Π_i^s returns $\perp$. Otherwise it proceeds with decryption as in Figure 6.

$\underline{\mathsf{Encrypt}(\Pi_i^s, m_0, m_1, h)}$	$\underline{\mathsf{Decrypt}(\Pi_i^s, c, h)}$
$(c^{(0)}, st_e^{(0)}) \leftarrow \mathsf{enc}(k_{\mathsf{enc}}^{\rho}, h, m_0)$	$(j,t) := \theta_i^s$
$(c^{(1)}, st_e^{(1)}) \leftarrow \mathsf{enc}(k_{\mathsf{enc}}^{\rho}, h, m_1)$	$v_i^s := v_i^s + 1$
if $c^{(0)} = \perp$ or $c^{(1)} = \perp$ **then return** $\perp$	**if** $b_i^s = 0$ **then return** $\perp$
$u_i^s := u_i^s + 1$	$(m, st_d) \leftarrow \mathsf{dec}(k_{\mathsf{dec}}^{\rho}, h, c, st_d)$
$(c_i^s[u_i^s], st_e) := (c^{(b_i^s)}, st_e^{(b_i^s)})$	**if** $v_i^s > u_j^t$ or $c \neq c_j^t[v_i^s]$, **then** $\mathsf{phase} := 1$
return $c_i^s[u_i^s]$	**if** $\mathsf{phase} = 1$ **then return** m

Fig. 6. Encrypt and Decrypt queries

We define the following game $\mathbf{Exec}_{\Pi}^{\mathsf{ACCE}}(\mathcal{A})$ between an adversary $\mathcal{A}$ and challenger $\mathcal{C}$:

1. The challenger $\mathcal{C}$, generates public/secret key pairs for each user $i \in I$ (by running $\mathcal{G}$) and returns the public keys to $\mathcal{A}$.
2. Adversary $\mathcal{A}$ is allowed to make as many $\mathsf{Send}^{\mathrm{pre}}$, Reveal, Corrupt, Encrypt, Decrypt queries as it likes.
3. Finally $\mathcal{A}$ outputs a triple (i, s, b').

We say the adversary $\mathcal{A}$ wins if it outputs $b' = b_i^s$. In this case the output of $\mathbf{Exec}_{\Pi}^{\mathsf{ACCE}}(\mathcal{A})$ is set to 1. Otherwise the experiment returns 0. Formally we define the advantage of $\mathcal{A}$ as

$$\mathbf{Adv}_{\Pi}^{\mathsf{ACCE}}(\mathcal{A}) = |\Pr[\mathbf{Exec}_{\Pi}^{\mathsf{ACCE}}(\mathcal{A}) = 1] - 1/2| = |\Pr[b' = b_i^s] - 1/2|.$$

Definition 22 (ACCE). *A protocol $P = \{\Pi, \mathcal{G}\}$ is a (t, ϵ)-**secure ACCE protocol** if for all adversaries $\mathcal{A}$ running in time t the following conditions hold (where $\epsilon = \epsilon_{EA} + \epsilon_{sAE}$):*

1. *(Entity Authentication/EA): There exists with probability at most ϵ_{EA} an oracle Π_i^s such that:*
   - *Π_i^s accepts when $\mathcal{A}$ issues its τ_0-th query with partner j, and*
   - *P_j is uncorrupted with $\tau_0 < \tau_j$ (i.e. at time of accept), and*
   - *$\mathcal{A}$ did not issue a Reveal-query to oracle Π_j^t, such that Π_j^t accepted while having a matching conversation to Π_i^s (if such an oracle exists).*
   - *there is no unique oracle Π_j^t such that Π_i^s has a (wire) matching conversation with Π_j^t.*
2. *(Secure Channel/sAE): When $\mathcal{A}$ terminates and outputs (i, s, b') such that*
   - *Π_i^s accepts when $\mathcal{A}$ issues its τ_0-th query with intended partner j, and*
   - *P_j is uncorrupted with $\tau_0 < \tau_j$ (i.e. at time of accept), and*
   - *$\mathcal{A}$ did not issue a Reveal-query to Π_i^s nor Π_j^t (such that they had a (wire) matching conversation).*

   the advantage is bounded by $\mathbf{Adv}_{\Pi}^{\mathsf{ACCE}}(\mathcal{A}) = |\Pr[b' = b_i^s] - 1/2| \leq \epsilon_{sAE}$.

C Previous Models for Secure Channels

C.1 Canetti–Krawczyk

The first attempt to combine the notions of secure key exchange and secure channels was made by Canetti and Krawczyk [8]. Here we shall highlight some of the similarities and differences with our new definitions.

Canetti and Krawczyk define a generic network channels protocol built upon a key exchange scheme and two generic functions *send* and *receive*. Here *send* would take as input some application message and output a message for the channel, *receive* would take as input a channel message and output an application message. The functions *Send* and *receive* may only be called after the key-exchange protocol has been completed, as a result [8] does not take into account protocols where their exists a key-confirmation step where messages are sent over the channel using the send functions. Not only does this create problems in defining protocol execution but it means no scheme of this type can be secure in their model. To facilitate a more modular security analysis Canetti and Krawczyk's approach is to first analyse the key-exchange protocol on its own using a notion for session-key security based on that of Bellare and Rogaway [4]. As a result the model is no longer suitable for analysing protocols with a key-confirmation step which uses the establish session key, as this would allow an adversary to trivially break security. In our model we also define a generic channels protocol but we shall consider protocols which have a key confirmation step utilising the session key.

To analyse the protocol as a whole, Canetti and Krawczyk split security into two parts. To be a secure channel protocol, a protocol must be both a secure encryption protocol and a secure authentication protocol. We will also take this approach, as it provides a more general framework. In some situations we may only require an authenticated channel thus having a separate definition for this can prove very useful.

Finally we discuss how Canetti and Krawczyk choose to handle receive (decryption) queries within their security models. To analyse secure encryption protocols they use an indistinguishability based notion, where the adversary is provided access to a left-or-right 'encryption' oracle. As a result an adversary should not be able to see the output of a receive call for a message for one participant which was previously output by a send call to his partner. The model therefore restricts by stating that if a plaintext output by *receive* was equal to a previous query to *send*, then this is not returned to the adversary. In our model we make a similar restriction but utilise the state of the encryption and decryption schemes to compare the channel messages output at different times during the protocol run. Canetti and Krawczyk justify their restriction by arguing that comparing the channel messages is overly constrained. Consider the header fields of a network protocol. In particular, the *time-to-live* field is decreased at ever router hop when it travels across a network. Therefore when the message is finally delivered it differs from that originally sent, despite the underlying plaintext message remaining the same. We state that our models can be easily extended to consider protocols with these types of header field by considering equivalence classes of channel messages.

C.2 ACCE Definition of Jager et al.

As mentioned in the introduction Jager et al. [12] combine the notions of authenticated key exchange [4, 5] and LHAE security [17] to give a combined notion of secure channel establishment. In this section we identify some issues with the approach of Jager et al. [12]. Our analysis is not concerned with the length hiding properties used by Jager et al. [12] and Paterson et al. [17] so we omit this aspect and consider only stateful authenticated encryption (sAE).

Reveal Queries We begin with what we argue is the main problem with their definition; namely at what point a Reveal query should be permitted. Reveal queries model unintended leakage of session keys from a participant. Security in the presence of Reveal queries then assures that keys which leak from one session do not affect the security of other sessions. Traditionally, Reveal queries are allowed once a participant has accepted a key. In both the new EMV scheme that we consider (cf. Section 2) and TLS (as considered by Jager et al.), the final part

of the key-exchange protocol involves a key confirmation step prior to a key being accepted. Here a message encrypted under the newly established session key is used to perform the final authentication of the sender and confirmation of the key. But if a session key is used prior to being accepted it seems logical that a Reveal should therefore be permitted as soon as keys are derived.

In Jager et al.'s definition they assume that "$\kappa \neq \emptyset$ if and only if $\delta = \texttt{accept}$" while in reality TLS has used κ prior to acceptance in order to encrypt both message m_{11} sent from client to server and message m_{13} sent from server to client. If instead, we allow the adverasary to Reveal as soon as a key is derived then we would be able to perform the following "attack":

- The client outputs the encrypted message m_{11}.
- The adversary reveals the client's key (which is allowed, as the client has derived the key).
- The adversary decrypts m_{11} and then re-encrypts it with new randomness, using the revealed key.
- Finally, the adversary forwards the new ciphertext to the server.
- The server accepts since the decrypted plaintext has not changed.

As a result the client and server will no longer have had a matching conversation. This is a requirement of the ACCE security definition and thus, TLS (and similarly EMV) cannot be proved secure with respect to this definition.

We note that Jager et al. [12] issued a revised version of their paper [11] which alters the definition of ACCE to prevent a similar issue with respect to the message m_{13}. In the first part of the definition they give the following additional restriction:

> *"$\mathcal{A}$ did not issue a Reveal-query to oracle Π_j^t, such that Π_j^t accepted while having a matching conversation to Π_i^s (if such an oracle exists)."*

With this restriction (Jager et al.'s description of) TLS can now be proved secured with relation to ACCE when reveals are only permitted once a key is *accepted*. But we stress that this model still fails to consider attacks of the form which we describe above, when Reveal queries are permitted as soon as a key has been *derived*. The point is that above, the client has been revealed but has yet to reach an accept state and so does *not* violate the new restriction. The adversary succeeds because the server has accepted without having a matching conversation with the client. In our new definition we shall permit reveal queries as soon as keys are derived, thus capturing all forms of this "reveal" attack. However, this does not mean there is an attack against TLS only that TLS has not been proved secure in this stronger security model.

Channel Messages In practice there are two types of messages sent over the wire during secure channel establishment and use. The first type of message that may be observed will be those used to establish the key. These are then followed by (encrypted) messages sent over the newly established secure channel. An adversary observing such a channel will not necessarily know when messages cease to be part of the key-exchange and become those of the secure channel. Let us consider the situation when an adversary tries to imitate a secure channel message. If a key has yet to be accepted then this message will affect the operation of the key exchange protocol.

The definition of Jager et al. allows the adversary to make three different types of query $\mathsf{Send}^{\mathrm{pre}}$, Encrypt and Decrypt each of which deals with a different type of message. $\mathsf{Send}^{\mathrm{pre}}$ is used only for messages sent as part of the key-exchange. The Encrypt and Decrypt operations will always return an error unless a key has been accepted. But in practice an adversary may not know when an oracle reaches an accept state. Consider the situation where an adversary makes a Decrypt call prior to a key being accepted. The input to both $\mathsf{Send}^{\mathrm{pre}}$ and Decrypt should model messages which have been received on the channel. In Jager et al.'s model an error would immediately be returned by the decryption oracle since no key has been accepted but in reality the message would actually interact with the current state of the key-exchange protocol. It is therefore intuitively more apealing to have a single Send operation which handles both the key-exchange and decryption operations depending on the state of the participant.

Thus, to achieve greater generality and mirror practice more effectively we shall resort to only using a single Send query in our model. When calling Send an adversary will specify a message m and a message type, type. Where the message type is either an application message (app) or a channel message (ch). Prior to the completion of the key exchange the operation will be ignored and the message will become part of the key-exchange execution. In addition our definition also allows the channel to have other capabilities (operations) such as sign not previously captured by the aforementioned definition.

D Key Secrecy

We define the following game $\mathbf{Exec}_{\Pi}^{\mathsf{KSec}}(\mathcal{A})$ between an adversary $\mathcal{A}$ and challenger $\mathcal{C}$:

1. The challenger $\mathcal{C}$, generates public/secret key pairs for each user $i \in I$ (by running $\mathcal{G}$) and returns the public keys to $\mathcal{A}$.
2. Adversary $\mathcal{A}$ is allowed to make as many NewSession, Send, Reveal, Corrupt queries as it likes.
3. Finally $\mathcal{A}$ outputs a pair Π^* and κ^*.

We say the adversary $\mathcal{A}$ wins if $\mathsf{F}(\Pi^*) = \mathsf{true}$ and κ^* is the key agreed by Π^*. In this case the output of $\mathbf{Exec}_{\Pi}^{\mathsf{Test}}(\mathcal{A})$ is set to 1. Otherwise the output is 0. We define the advantage of $\mathcal{A}$ to be

$$\mathbf{Adv}_{\Pi}^{\mathsf{KSec}}(\mathcal{A}) = |\Pr[\mathbf{Exec}_{\Pi}^{\mathsf{KSec}}(\mathcal{A}) = 1]|.$$

Definition 23. *(Key Secrecy) Protocol $P = \{\Pi, \mathcal{G}\}$ is a $(t, \epsilon_{\mathsf{KSec}})$-**key secret AK protocol** if for all adversaries $\mathcal{A}$ running in time t the following holds:*

1. *In the presence of a benign adversary on Π_i^s and Π_j^t both oracles accept holding the same session identifier sid, the same session key κ, and this key is distributed uniformly at random on $\{0,1\}^k$.*
2. *$\mathcal{A}$'s advantage is bounded by $\mathbf{Adv}_{\Pi}^{\mathsf{KSec}}(\mathcal{A}) \leq \epsilon_{\mathsf{KSec}}$.*

We can also define a weaker version of this model for one-sided authentication by running the experiment in the same way as before but changing the winning condition slightly. We say the adversary $\mathcal{A}$ wins the wKSec experiment if $\mathsf{OSF}(\Pi^*) = \mathsf{true}$ and κ^* is the key agreed by Π^*.

Definition 24. *(Weak Key Secrecy) A protocol $P = \{\Pi, \mathcal{G}\}$ is a $(t, \epsilon_{\mathsf{wKSec}})$-**weak Key-secure AK protocol** if for all adversaries $\mathcal{A}$ running in time t the following holds:*

1. *In the presence of a benign adversary on Π_i^s and Π_j^t both oracles accept holding the same session identifier sid, the same session key κ and this key is distributed uniformly at random on $\{0,1\}^k$.*
2. *$\mathcal{A}$'s advantage is bounded by $\mathbf{Adv}_{\Pi}^{\mathsf{wKSec}}(\mathcal{A}) \leq \epsilon_{\mathsf{wKSec}}$.*

E Proof of Theorem 2

Proof. We shall prove this result via a sequence of games. Let $\mathcal{A}$ be adversary attacking Π in the sense of unlink.

Game 0: This game is identical to $\mathbf{Exec}_{\Pi}^{\mathsf{unlink}}(\mathcal{A})$.

$$\Pr[\mathsf{Game0} \Rightarrow 1] - \frac{1}{2} = \mathbf{Adv}_{\Pi}^{\mathsf{unlink}}(\mathcal{A}).$$

Game 1: The challenger now selects at random i_0^* and i_1^*. The game aborts and returns random b' if $\mathcal{A}$ does not output $i_0 = i_0^*$ and $i_1 = i_1^*$. We obtain

$$\Pr[\mathsf{Game0} \Rightarrow 1] - \frac{1}{2} \leq n_C^2 \cdot \left(\Pr[\mathsf{Game1} \Rightarrow 1] - \frac{1}{2}\right).$$

Game 2: This proceeds identically to the previous game but aborts if $\mathcal{A}$ makes a query to H which reveals the key for the oracle $\mathcal{O}$. We obtain

$$\Pr[\mathsf{Game1} \Rightarrow 1] \leq \Pr[\mathsf{Game2} \Rightarrow 1] + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{B}').$$

Game 3: This proceeds identically to the previous game except that whenever Send is called with $\mathcal{O}_C$ and $\mathtt{type} = \mathsf{app}$ then the challenger replaces m with a random message which it then encrypts. Again it is easy to see that we obtain

$$\Pr[\mathsf{Game2} \Rightarrow 1] \leq \Pr[\mathsf{Game3} \Rightarrow 1] + \mathbf{Adv}_{\mathsf{AE}}^{\mathsf{indsfcca}}(\mathcal{C}).$$

It remains to study the probability that $\mathcal{A}$ wins ($\mathsf{Game2} \Rightarrow 1$). Since ciphertexts are now distributed uniformly at random the only useful information that $\mathcal{A}$ can determine are the public keys $Q_{i_0^*}$, $Q_{i_1^*}$, and the blinded challenge value $aQ_{i_b^*}$. Since a is chosen at random from $\mathbb{F}_q$, then the distributions $(Q_{i_0^*}, Q_{i_1^*}, aQ_{i_0^*})$ and $(Q_{i_0^*}, Q_{i_1^*}, aQ_{i_1^*})$ are identical, i.e. the advantage is zero even if the adversary is computationally unbounded. We therefore have:

$$\Pr[\mathsf{Game3} \Rightarrow 1] - \frac{1}{2} = 0.$$

Combining all of the above:

$$\begin{aligned}
\mathbf{Adv}_{\Pi}^{\mathsf{unlink}}(\mathcal{A}) &= \Pr[\mathsf{Game0} \Rightarrow 1] - \frac{1}{2} \\
&\leq n_C^2 \cdot \left(\Pr[\mathsf{Game1} \Rightarrow 1] - \frac{1}{2} \right) \\
&\leq n_C^2 \cdot \left(\Pr[\mathsf{Game2} \Rightarrow 1] - \frac{1}{2} + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{B}') \right) \\
&\leq n_C^2 \cdot \left(\Pr[\mathsf{Game3} \Rightarrow 1] - \frac{1}{2} + \mathbf{Adv}_{\mathsf{AE}}^{\mathsf{indsfcca}}(\mathcal{C}) + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{B}') \right) \\
&\leq n_C^2 \cdot \left(\mathbf{Adv}_{\mathsf{AE}}^{\mathsf{indsfcca}}(\mathcal{C}) + \mathbf{Adv}_{\pi}^{\mathsf{wKSec}}(\mathcal{B}') \right) \\
&\leq n_C^2 \cdot \left(\mathbf{Adv}_{\mathsf{AE}}^{\mathsf{indsfcca}}(\mathcal{C}) + n_C \cdot (1 - 1/|h|) \cdot \mathbf{Adv}_{E(\mathbb{F}_p)}^{\mathsf{Gap\text{-}DH}}(\mathcal{B}) \right).
\end{aligned}$$

□

We note that if we were permitted to have a small (as would be the case in the original EMV proposal) distinguishing the two distributions $(Q_{i_0^*}, Q_{i_1^*}, aQ_{i_0^*})$ and $(Q_{i_0^*}, Q_{i_1^*}, aQ_{i_1^*})$ may no longer be hard. Let l denote the maximum bit length of a. The real question of interest would then be how small can l be before the above problem becomes easy for computationally bounded adversaries. It is clear that the best attack against the problem for $2^l \ll q$ will be Pollard Lambda method [18], which runs in time $O(2^{l/2})$. This implies that a 32-bit randomizer a only gives 16-bits of security and an 80-bit randomizer only gives 40 bits of security.