Unpicking PLAID

A Cryptographic Analysis of an ISO-standards-track Authentication Protocol

Jean Paul Degabriele[1] Victoria Fehr[2] Marc Fischlin[2] Tommaso Gagliardoni[2]
Felix Günther[2] Giorgia Azzurra Marson[2] Arno Mittelbach[2] Kenneth G. Paterson[1]

[1] Information Security Group, Royal Holloway, University of London
[2] Cryptoplexity, Technische Universität Darmstadt, Germany
{j.p.degabriele,kenny.paterson}@rhul.ac.uk, marc.fischlin@cryptoplexity.de,
{victoria.fehr,tommaso.gagliardoni,giorgia.marson,arno.mittelbach}@cased.de,
guenther@cs.tu-darmstadt.de

Abstract. The Protocol for Lightweight Authentication of Identity (PLAID) aims at secure and private authentication between a smart card and a terminal. Originally developed by a unit of the Australian Department of Human Services for physical and logical access control, PLAID has now been standardized as an Australian standard AS-5185-2010 and is currently in the fast track standardization process for ISO/IEC 25185-1.2. We present a cryptographic evaluation of PLAID. As well as reporting a number of undesirable cryptographic features of the protocol, we show that the privacy properties of PLAID are significantly weaker than claimed: using a variety of techniques we can fingerprint and then later identify cards. These techniques involve a novel application of standard statistical and data analysis techniques in cryptography. We also discuss countermeasures to our attacks.

Keywords. Protocol analysis, ISO standard, PLAID, authentication protocol, privacy

Contents

1 Introduction

PLAID, the Protocol for Lightweight Authentication of Identity, is a contactless authentication protocol intended to be run between terminals and smartcards. The protocol was designed by Centrelink, an agency of the Australian government's Department of Human Services (DHS). According to the developers it is supposed to provide a cryptographically strong, fast, and private protocol for physical and logical access control, without exposing "card or cardholder identifying information or any other information which is useful to an attacker" [Cen09, Aus10, ISO14].

PLAID was initially proposed for use in the internal ID management of Centrelink [Kli10]. However, the intended scope of applications has since significantly broadened to include the whole of DHS and the Australian Department of Defence [Tay12]. Indeed, the protocol's promoters aspire to broader commercial and governmental deployment, including on an international level [Dep14]. Strategies that are mentioned to support these aspirations include freely available intellectual property and outreach to other governmental organizations. To the latter end, NIST organized a workshop to explore the potential of PLAID for U.S. Federal Agencies in July 2009 [Nat09].

Another strategy that is being actively pursued is standardization. PLAID was previously registered as the Australian standard AS-5185-2010 [Aus10] and was then entered into the ISO/IEC standardization process via the fast track procedure. At the time of writing, the current ISO/IEC version is draft international standard (DIS) 25185-1.2 [ISO14] and is currently in the "Enquiry phase" 40.60 (close of voting). Minor changes in the original protocol to match the international standard have been applied. Reference implementations, based on the Australian standard, are available both from the Australian Department of Human Services (in Version 8.04) and of the Australian Department of Defence (in draft version 1.0.0).

The protocol. The main aim of the protocol is to perform mutual authentication and establish a shared key between the terminal (IFD) and the card (ICC). To this end the terminal and the card exchange nonces RND1 and RND2 in encrypted form and then derive the session key as part of the hash value of the two nonces. Encryption here uses both asymmetric RSA-based encryption (when the card transmits RND1 to the terminal) and symmetric AES-based encryption (when the terminal sends RND2 to the card). Authentication of the partner is presumably guaranteed by the fact that a party should know the secret key in order to be able to decrypt the other party's nonce. An overview of the protocol is depicted in Figure 1, where the encrypted nonces are exchanged with transmissions ^{e}STR1 and ^{e}STR2. The card confirms the receipt of RND2 by sending a string encrypted under the derived key in ^{e}STR3.

The role of the terminal's initial message KeySetIDs is as follows. Each PLAID deployment involves a set of key pairs consisting of an RSA key and an AES key. Each terminal and each card stores a certain subset of these pairs. More precisely, each terminal holds a set of RSA key pairs (both encryption and decryption key) and corresponding AES master keys, while each card holds a set of RSA public keys and card-specific AES keys, derived from the corresponding AES master keys using a card identity. The keys held by a card are intended to control what types of access the card should have, so each key represents a capability. The actually deployed pair of keys is negotiated during the protocol itself, by having the terminal send a sequence of supported RSA key identifiers KeySetID in the first message. Even though the encryption key in RSA is usually public, in PLAID it is kept secret to enhance privacy (since, for example, the set of RSA keys held by a card could be used to identify the card and track its use in a deployment).

One distinctive feature of the protocol, added for privacy reasons, is that the card switches to using a pair of so-called shill keys in case of an error. That is, if the card detects some potential error, then it uses its card-specific RSA shill key and AES shill key to encrypt random data. This mechanism is intended to hide information about failures from an adversary and thereby prevent leakage about which keys are possessed by a particular card.

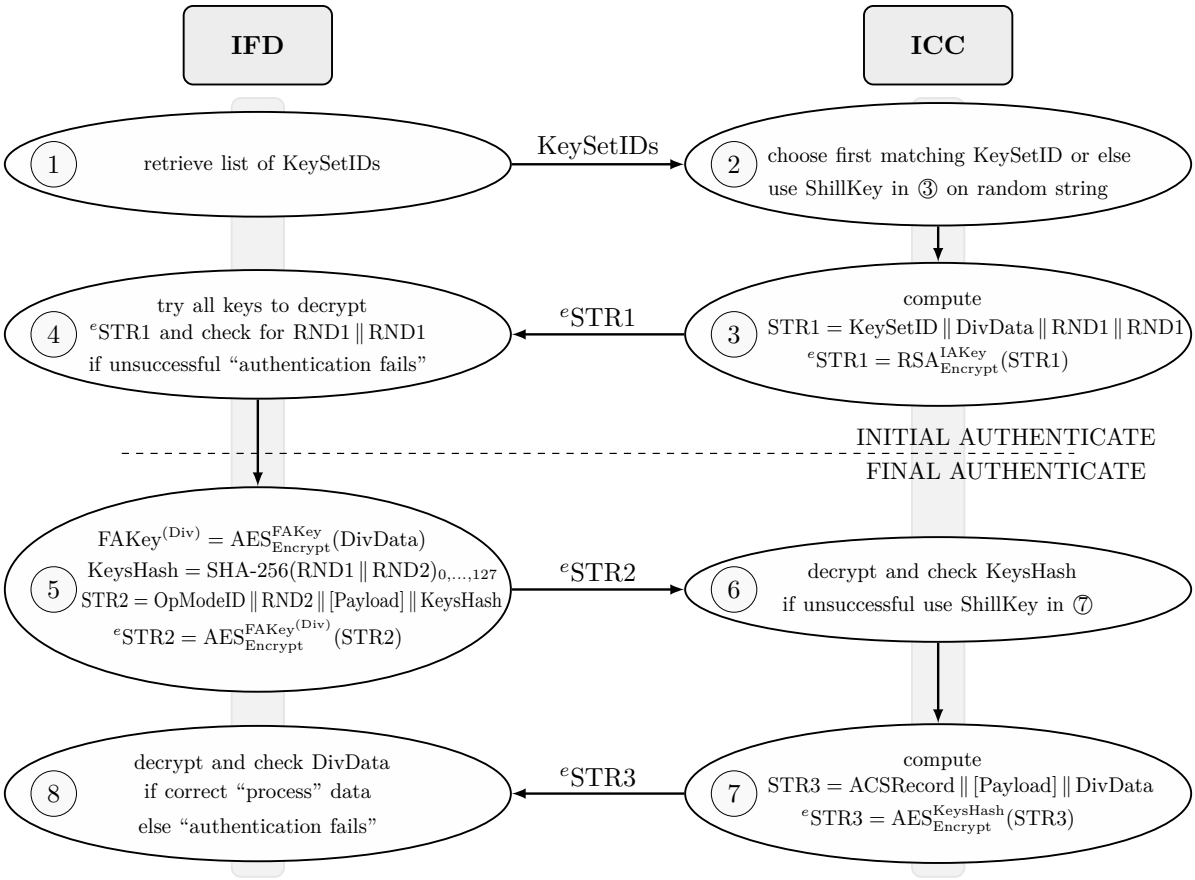

Figure 1: PLAID protocol overview.

Previous security analyses. Centrelink's accompanying description of PLAID [Cen09] claims that PLAID is highly resistant against leakage of card or card holder identifying information, against various forms of active attacks, and provides mutual authentication. The document states as a goal that the protocol shall be "evaluated by the most respected cryptographic organisations, and the broader cryptographic community." For version 8 the document [Cen09] refers to the input by various agencies like NIST and of "a number of independent cryptographic experts and consultants, a number of respected commercial cryptographic teams, as well as the internal Centrelink team."

However, we are not aware of any publicly available cryptographic evaluation of PLAID. None of the claimed security properties is backed up by arguments, nor matched against more precise formalizations in the description [Cen09] or standards [Aus10, ISO14]. Some useful comments about the protocol's security have been given by the national representatives in the previous version of the ISO standard and documented there [ISO13]. These comments refer partly to the points discussed in Section 5, where we asses them in a cryptographic context.

PLAID has been scrutinized to some extent by using formal methods and automated tools. Watanabe [Wat13], using Scyther, and Sakurada [Sak13], using ProVerif, confirmed that PLAID satisfies some form of mutual authentication and some level of secrecy of the session key, assuming idealized cryptographic primitives. It remained unclear to us what this assurance means in a cryptographic sense. Neither of the works considers privacy aspects.

Finally, the Master's thesis of Kiat and Run [KR12] at the Naval Postgraduate School compares PLAID with a similar protocol, the ANSI/INCITS 504-1-2013 standard OPACITY. The conclusion is indecisive

and is primarily based on deployment characteristics. The authors evaluate cryptographic properties only on a superficial level. Indeed, while the thesis does not pinpoint at any major weakness in OPACITY, a cryptographic analysis [DFG+13] was less positive.

Our results. According to the developers of PLAID, the lack of privacy in previous efforts was one of the main reasons to introduce a new authentication protocol [Ris09]. Indeed, PLAID is described as highly resistant against "the leakage of individually identifiable, unique or determinable data or characteristics of the smart card or the holder during authentication." [Cen09]. We argue here that PLAID does not achieve this ambitious goal. More precisely, we describe and evaluate a suite of attacks that break the privacy goals of PLAID, enabling cards to be efficiently identified in a number of realistic scenarios. We also identify some countermeasures to our attacks.

In more detail, two of our attacks exploit PLAID's use of shill keys, which, being card-specific, can serve as a proxy for the card identity. While the shill keys themselves are not transmitted in the protocol, we show how they can be statistically estimated from RSA ciphertexts observed in protocol runs, enabling cards to be initially fingerprinted and then later re-identified. These two "shill key fingerprinting" attacks, presented in Section 3, deploy different techniques to perform the statistical estimation, and apply in different attack scenarios. Our first attack uses the standard solution to what is known as the "German Tank Problem", which concerns estimating the maximum of a discrete uniform distribution from a number of samples, while our second attack uses clustering techniques (and in particular the standard k-means clustering algorithm) to perform the estimation of the shill keys.

Our third attack, targeting the terminal's initial message KeySetIDs, is called "keyset fingerprinting" and is presented in Section 4. It exploits specific properties of the protocol flow to extract information about the set of keys held by a given card, potentially allowing us to draw conclusions about the card holder (e.g., via access authorizations). We show that this information can be efficiently extracted by interacting with a card a number of times and observing how the protocol proceeds (or fails to proceed). The information obtained in this attack may already be sufficient to identify individual cards from amongst a population, depending on the exact characteristics of a given deployment. The attack can also be combined with our first two attacks to increase their efficiency (by reducing the number of possible keys that need to be considered in the re-identification phase).

Finally, in Section 5, we make a number of other observations on cryptographic aspects of the PLAID protocol, focusing in particular on its lack of forward security, the use of weak RSA encryption, the lack of integrity protection for the symmetric encryption and a number of imprecisions in its specification. Some of the issues have already been briefly touched upon in the national body comments on the previous ISO standard version [ISO13]; some aspects, like the lack of forward security, are new.

Reaction of the Responsible Authorities. We communicated our results to both the ISO 25185-1 project editor and to a contact person at the Department of Human Services. They are currently looking into our findings. The ISO project editor Graeme Freedman pointed out to us [Fre14] that card-identifying information may be also available by other means, such as through the so-called Card Production Life Cycle (CPLC) data. The CPLC data contain information like serial numbers and manufacturers, uniquely identifying cards on a global scale. For privacy reasons access to the CPLC data must thus be restricted for PLAID. Indeed, the ISO draft standard itself already mentions this issue: "Consider switching off access to administrative applications from contactless interfaces, particularly ones which store unique card identification information such as the GlobalPlatform Card Production Life Cycle (CPLC) data." [ISO14]. Our results show, however, that even if one restricts access to such administrative data, then PLAID still leaks card and cardholder identifying information.

Variable	Description
ACSRecord	An access-control system record for each operation mode required for authentication.
DivData	A "random or unique" 16-byte ICC identifier.
FAKey	A 16-byte AES key which can be seen as master key to compute the diversified key used in the protocol (only known to the IFD).
$\text{FAKey}^{(\text{Div})}$	A 16-byte AES key derived from the FAKey and used in the FA phase.
IAKey	A 2048-bit pre-shared RSA key pair used in the IA phase. The ICC only knows the public key part.
KeySetID	A 2-byte index value identifying an IAKey and FAKey or $\text{FAKey}^{(\text{Div})}$, respectively.
OpModeID	A 2-byte index value identifying the operation mode. This value indicates which ACSRecord and payload the ICC needs to provide for authentication.
RNDi	A 16-byte random string for $i = 1, 2$.
KeysHash[1]	A 16-byte session key computed by IFD and ICC used in the FA phase.
ShillKey	A pair of 2048-bit RSA public key and 16-byte AES key of the ICC (randomly chosen per ICC during setup). These keys are to be used instead of error messages to simulate the next step of the protocol camouflaging that something went wrong.

Table 1: Most important fields and identifiers of PLAID.

2 PLAID Protocol Description

In this section we give a detailed description of PLAID according to the specification of the draft ISO/IEC DIS 25185-1.2 [ISO14]. A more concise overview of the protocol flow is depicted in Figure 1. To make our description as close as possible to the original specification [ISO14] we denote terminal and card by IFD (Interface Device) and ICC (Integrated Circuit Card) respectively. Table 1 provides a summary of the most important fields and objects occurring in the protocol.

2.1 PLAID Setup

In the setup phase PLAID initializes both terminals (IFDs) and cards (ICCs). PLAID supports up to 2^{16} key sets, each consisting of an RSA key pair IAKey_i and an AES key FAKey_i. Each terminal and each card hold a subset of these overall possible key pairs, according to some access-control policy. However, the card only holds the public key part of the IAKey as well as a processed version of the original FAKey. More concretely, the card does not keep the FAKeys directly, but only a diversified version $\text{FAKey}^{(\text{Div})} = \text{AES}^{\text{FAKey}}_{\text{Encrypt}}(\text{DivData})$, where DivData is a 128-bit card identifier. The standard [ISO14] highlights that these diversification data should be "random or unique". Using the diversified key instead of FAKey should retain security for other cards, in the case of a card being compromised and hence some of the (diversified) keys are disclosed. In addition to the RSA keys, $\text{FAKey}^{(\text{Div})}$ and the value DivData, each card receives a pair of individual distress-keys (called ShillKey): a random encryption RSA key and a random AES key. These "shill keys" should be used to encrypt random data in case an error is detected, thus camouflaging errors or de-facto aborts on the card.

2.2 Initial Authenticate

The IA phase aims at exchanging the necessary information to compute the symmetric keys used in the FA phase as well as transferring DivData, the card-specific data later needed to guarante authenticity of the final message, securely to the terminal.

[1]Note that in the original draft KeysHash refers to the entire 32 byte output of SHA-256(RND1 $\|$ RND2) and the term *session key* is used to refer to the first 16 bytes which are used as secret key in the final message. For simplicity we refer to the session key as KeysHash in this paper.

Step 1 (IFD) – IA Command: The interaction is initiated by the IFD, which transmits the complete sequence of supported KeySetIDs (in order of preference) to the ICC.

Step 2 (ICC) – IA Command Evalution: Upon receiving a set of KeySetIDs, the ICC traverses the entire list of indices to find the first KeySetID it supports, which determines the IAKey for RSA encryption. To prevent timing attacks it does not abort the search, even if a match has occurred. If no match is found, in Step 3 the ICC will encrypt a randomly generated string using its ShillKey.[2]

Step 3 (ICC) – IA Response: The ICC generates RND1, retrieves its DivData and derives string STR1, together with an encryption of it under IAKey, as follows:

$$\mathrm{STR1} = \mathrm{KeySetID} \,\|\, \mathrm{DivData} \,\|\, \mathrm{RND1} \,\|\, \mathrm{RND1}, \quad {}^{e}\mathrm{STR1} = \mathrm{RSA}^{\mathrm{IAKey}}_{\mathrm{Encrypt}}(\mathrm{STR1}) \ .$$

The encrypted string ^{e}STR1 is sent to the IFD. Here PKCS#1 v1.5 padding is used.

Step 4 (IFD) – IA Response Evaluation: The IFD trial-decrypts ^{e}STR1 with all possible private IAKeys indexed by its KeySetID list and, for each valid decryption, it checks if the last two 16-byte blocks are equal. Again, to prevent timing attacks the IFD will continue the search even if a matching string has already been found. The (first) match is then used to extract KeySetID[3], DivData, and RND1. If no plaintext is of the anticipated format, authentication fails.[4]

2.3 Final Authenticate

The FA phase permits to specify the operation mode and to exchange data, like a PIN or biometrics, needed to complete the authentication. Here the diversified key $\mathrm{FAKey}^{(\mathrm{Div})}$ (stored on the card and previously computed by the terminal during the IA phase) and a derived session key are used to secure the communication. The card authenticates by proving its ability to decrypt ^{e}STR2 as well as to include the correct DivData (transmitted in the previous IA phase) in the final message ^{e}STR3.

Step 5 (IFD) – FA Command: The IFD generates the 16-byte nonce RND2 and computes the unique session key KeysHash as the first 128 bits of

$$\text{SHA-256}(\mathrm{RND1} \,\|\, \mathrm{RND2}) \ .$$

Next, using the master FAKey indexed by KeySetID, it computes the diversified AES key

$$\mathrm{FAKey}^{(\mathrm{Div})} = \mathrm{AES}^{\mathrm{FAKey}}_{\mathrm{Encrypt}}(\mathrm{DivData}) \ ,$$

which corresponds to the AES key stored on the ICC under index KeySetID. The latter is used to encrypt

$$\mathrm{STR2} = \mathrm{OpModeID} \,\|\, \mathrm{RND2} \,\|\, [\mathrm{Payload}] \,\|\, \mathrm{KeysHash} \ ,$$

using AES in CBC mode with the all-zero string as initialization vector, where Payload is an optional, variable-size field that depends on the operation mode. Concerning padding, the standard refers to the ISO/IEC 9797-1 method 2, where one byte `0x80` is appended, followed by blocks of `0x00` bytes until the length is a multiple of the block length.[5] The resulting string

$${}^{e}\mathrm{STR2} = \mathrm{AES}^{\mathrm{FAKey}^{(\mathrm{Div})}}_{\mathrm{Encrypt}}(\mathrm{STR2}) \ ,$$

is then transmitted to the ICC.

[2]The standard neither specifies the exact format nor the length of this randomly generated string.

[3]The standard is ambiguous in whether the trial KeySetID of the IFD or the value contained in ^{e}STR1 is stored.

[4]The standard does not specify what is meant by "authentication fails." We assume the protocol aborts in this case.

[5]Though referring to ISO/IEC 9797-1 method 2, the standard explicitly describes a different padding method and thus makes unambiguous decoding impossible (cf. Section 5.4).

Step 6 (ICC) – FA Command Evaluation: The ICC decrypts ${}^{e}\text{STR2}$ with $\text{FAKey}^{(\text{Div})}$ and retrieves RND2. It computes the session key as described above as first half of SHA-256(RND1 $\|$ RND2) and compares the result to the value KeysHash extracted from the decrypted ${}^{e}\text{STR2}$. If they do not match the ICC encrypts a random byte string[6] using its AES ShillKey in the FA Response. Else Payload, if given, should be processed as specified by the implementation.

Step 7 (ICC) – FA Response: The ICC retrieves the Payload data specified by the operation mode (if necessary) and encrypts

$$\text{STR3} = \text{ACSRecord} \,\|\, [\text{Payload}] \,\|\, \text{DivData} \ ,$$

using AES in CBC mode with the all-zero string as initialization vector. Again, Payload is an optional, variable-size field which may (and usually will) differ from the Payload in Step 5. The resulting ciphertext

$${}^{e}\text{STR3} = \text{AES}^{\text{KeysHash}}_{\text{Encrypt}}(\text{STR3}) \ ,$$

is transmitted as final message to the IFD.

Step 8 (IFD) – FA Response Evaluation: The IFD decrypts ${}^{e}\text{STR3}$ and checks whether the recovered DivData matches the one received in the IA phase: if so, then the other data is considered authenticated and processed according to the implementation, otherwise authentication fails.

3 ShillKey Fingerprinting – Tracing Cards in PLAID

According to the developers of PLAID, privacy was one of the main reasons to introduce a new authentication protocol. In this and the next section we present three attacks on the privacy of PLAID contradicting the claims that no static information is available to be exploited. In this section we focus on the *traceability* of cards, that is, we consider an adversary who learns some information about one or more cards and then tries to identify these cards at a later time.

We consider two distinct attack scenarios, each consisting of a fingerprinting phase and then an identification phase. The difference is roughly that in the first scenario the fingerprinting is a supervised learning phase in the sense that we can attribute execution traces to cards, whereas the second setting corresponds to unsupervised learning where we get a set of random traces. More precisely:

- In the first scenario we allow the adversary to first interact in turn with each and every card in the system in a number of protocol runs (the fingerprinting phase). We then draw a card at random and let the adversary interact with this specific card a number of times, with the adversary's goal being to identify which of the cards was selected. The adversary's ability to interact with each card in the system in turn in the identification is not wholly realistic. However, given the high success rates of this attack that we will report below, we believe that good success rates would still be achieved in the more realistic scenario where the adversary does not have the guarantee of being able to interact with each distinct card in turn in a first phase, but instead must build up its picture of the system as it goes along.

- In the second scenario, which is much more challenging for the adversary, we do not allow the adversary to interact in turn with every card in a number of protocol runs, but simply present it with a sequence of transcripts of individual protocol executions, each execution involving a randomly chosen card. The identification phase and the adversary's goal are the same as before. This much

[6] Again, the standard does neither specify the exact format nor the length (note that STR3 in Step 7 contains a variable sized field Payload) of this random byte string.

more demanding attack scenario models a situation where the adversary cannot interact many times with each distinct card during fingerprinting, but only in one protocol run at a time with a random card.

In Section 4 we consider a different attack scenario and show how an adversary can learn the capabilities of a card (that is, it learns *which* keys are stored on a card). Besides being a serious breach of privacy on its own, this attack can also be combined with the attacks described in this section to gain better performance.

Our attacks in this section specifically target the ShillKey values used by PLAID. A ShillKey pair, generated for every card, contains an RSA public key and an AES key that are to be used in the IA and in the FA phases respectively in place of the actual keys should an error in the terminal message be detected. Intended as a security measure—to prevent attackers from exploiting potential information leaked by error messages—the use of the ShillKey turns out to drastically weaken the anonymity properties of PLAID.

Before explaining the details of the attacks, we note that in order to run the attacks in this section, we need to be able to force each card into replying with RSA ciphertexts generated using its ShillKey in the first phase of the protocol. This can, however, easily be arranged by sending the card a first message containing an empty sequence of KeySetIDs, or a set of KeySetIDs containing a single and particularly high index that is not in use in any card on the system. Thus we may assume that the adversary is able to gather samples of ShillKey ciphertexts from cards at will.

3.1 Tracing Cards via ShillKey Ciphertexts

We consider the following situation: we assume the system has t cards with corresponding ShillKey moduli $N_1, \ldots, N_t$, where each N_i is an n-bit RSA modulus (the current draft version gives $n = 2048$ [ISO14]). Our basic attack considers the following scenario. In a first phase the adversary learns, for every card in the system, k_1 encryptions of a random message under the card's ShillKey (N_j, e_j); then, in a challenge phase the adversary is given k_2 fresh ciphertexts (again for random messages) computed under ShillKey (N_{j^*}, e_{j^*}), for j^* chosen uniformly at random from $\{1, \ldots, t\}$. The adversary's goal is to identify from which card the challenge ciphertexts come, that is, to output the correct index j^*. We define the adversary's advantage as its success probability bounded away from the guessing probability $\frac{1}{t}$.

The idea behind our first attack is that, although each ShillKey (N_j, e_j) is meant to be kept private, each of the k_1 ciphertexts $X_{i,j}$ computed using (N_j, e_j) leaks some information about the modulus N_j. Specifically, we learn that $X_{i,j} < N_j$ for each i. Similarly, in the challenge phase, where we have k_2 ciphertexts computed using (N_{j^*}, e_{j^*}), each ciphertext leaks some information about the challenge ShillKey modulus. Starting from this observation, we now seek a procedure to obtain a good estimate of the ShillKey moduli given only a certain number of corresponding ciphertexts for each modulus.

The problem can be reposed as follows. Notice that each ciphertext $X_{i,j}$ can be regarded as a uniformly random integer in the range $[1, N_j - 1]$. We are then faced with the task of estimating N_j, which is one more than the size of the interval from which the sample comes. This is essentially an instance of a classical statistical problem that is known as the *German Tank Problem*[7]. A naive approach would be to use twice the mean values of the samples $X_{i,j}$ as an estimate for N_j. A statistically strictly better approach is to use as an estimator for N_j the value

$$\tilde{N}_j = m_j + \frac{m_j}{k_1} \ ,$$

where m_j is the maximum value of the observed samples $X_{i,j}$ and k_1 is the number of samples. It basically corresponds to the maximum plus the average distance of observed samples. This estimator arises from a

[7] See [Joh94] for a good introduction. The name stems from the problem initially being posed as that of estimating the total number of tanks in the German army from observing a subset of their serial numbers.

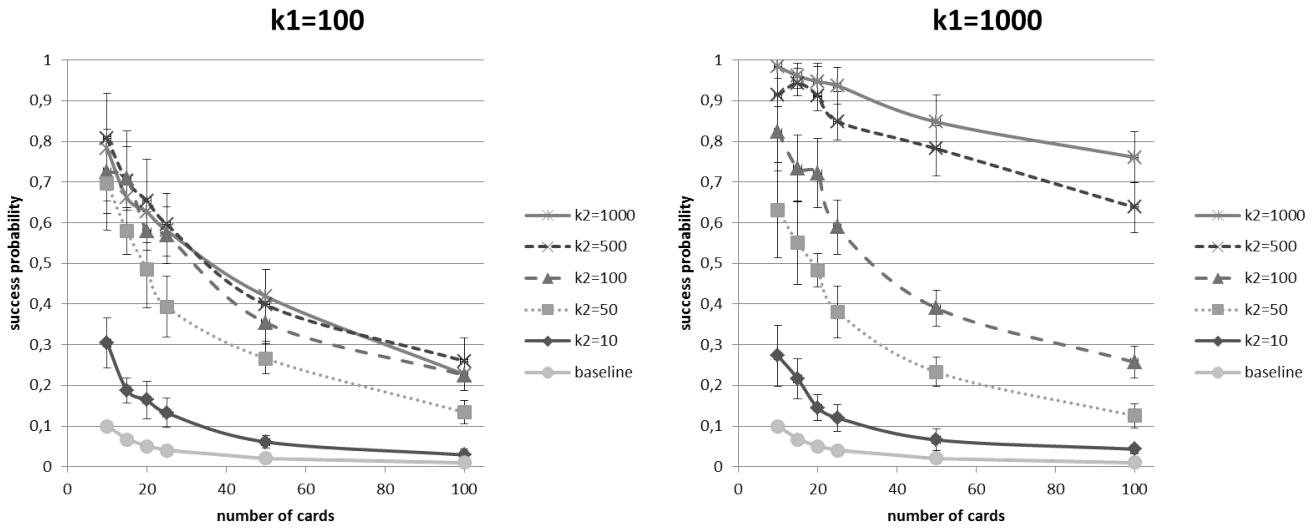

Figure 2: Simulations of the ShillKey attack. On the left with k_1 (the number of samples during the fingerprinting phase) set to 100 and on the right with k_1 set to 1000. The success probability is averaged over ten runs with one hundred repetitions of the identification phase and the simulation was done with $t = 10, 15, 20, 25, 50, 100$ cards. The baseline indicates the success probability of an adversary that tries to win the game by purely guessing.

frequentist interpretation of the problem, and has the benefit of providing what is known as a Minimum-variance Uniform Estimator (MVUE). It can be replaced by a more appropriate Bayesian estimator, but the estimator above is sufficient for our purposes.

Our first attack proceeds using this estimator as follows. In the first phase, we use it to produce estimates $\tilde{N}_j$ for each of the ShillKey moduli N_j. In the challenge phase, we again use it to produce an estimate $\tilde{N}^*$ for the challenge ShillKey modulus (now with parameter k_2, representing the number of samples available in that phase). We finally output as our guess for the challenge index j^* the index j for which $\tilde{N}^*$ is closest in absolute value to $\tilde{N}_j$, that is,

$$\arg\min_j \left| \tilde{N}^* - \tilde{N}_j \right| \quad .$$

This concludes the description of our first attack.

Simulation results. We have conducted extensive simulations of the first attack above for various values of t (the number of cards), k_1 (the number of ciphertext samples per card available in the first phase) and k_2 (the number of ciphertext samples in the challenge phase). A selection of our results are depicted in Figure 2.

It can be seen that even with many cards in the system ($t = 100$, say), our attack significantly outperforms simply guessing the card's identity. This is particularly so when the number of samples k_2 available in the challenge phase is large. However, even with k_2 as low as 50, the attack performance is still significantly better than guessing; given the target execution time of the PLAID protocol of 300 ms, this many samples could be gathered from a card within 15 s.

It is also the case that increasing the number of samples available in the first phase of the attack, k_1, improves the attack performance (compare the left and right sub-figures in Figure 2, which correspond to values of k_1 equal to 100 and 1000, respectively). This is particularly accentuated at higher levels of k_2, where the attack's success probability is better than 75%, even for a system with $t = 100$ cards.

3.2 Tracing Cards from a Mixed Set of ShillKey Ciphertexts

For the basic attack in Section 3.1, we assumed that during the initial phase the attacker was able to identify ciphertexts computed from the same key. In our second attack, we relax this assumption: we now give the attacker a large mixed set of $k_1 \times t$ ciphertext samples, each sample coming from a randomly selected card. The challenge phase of the attack proceeds as before where the attacker obtains a small sample of k_2 ciphertexts computed by the same card, and the attacker's goal is to identify this card.

The challenge now is to somehow process this mixed set of samples in order to extract reasonable estimates of the individual RSA moduli. We accomplish this by means of a heuristic clustering technique. Assuming that we know the number of cards t used to produce the mixed sample set, let $N_1, \ldots, N_t$ represent their ShillKey moduli in increasing order. From the mixed sample of ciphertexts we ignore all samples smaller than 2^{2047}. We then use a standard clustering technique based on the k-means algorithm to group the remaining ciphertext samples into t clusters approximating the intervals $[N_j, N_{j+1})$, for $j \in \{0, 1, \ldots, t\}$ and $N_0 = 2^{2047}$. Once we have this set of clusters, we then obtain an estimate for the ShillKey modulus N_{i+1} by using the German Tank estimator on the cluster corresponding to the interval $[N_i, N_{i+1})$.

We now describe the clustering algorithm in more detail. We initially assign to each of the t clusters a uniformly random value in the range $(2^{2047}, 2^{2048})$. This value is called the *centroid* of the cluster. For each ciphertext sample greater than 2^{2047} we calculate its distance from each of the cluster centroids, and assign that ciphertext to the cluster to whose centroid it is closest. The distance metric is merely the absolute value of the arithmetic difference. Once that every ciphertext sample has been assigned to a cluster we ensure that no cluster is empty. If an empty cluster is found, we pick another cluster at random whose size is greater than one and move its largest element to the empty cluster. We then set the centroid of each cluster to be the mean of the ciphertext samples contained in that cluster, as per the standard k-means algorithm. We iterate this process of assigning ciphertext samples to clusters and recalculating their centroids until the centroids converge to stable values, or the maximum number of iterations is exceeded.

In the challenge phase of the attack, the attacker is given k_2 ciphertexts computed by the same card. Here, our attack proceeds identically to the previous one: the attacker uses the estimator to produce an estimate $\tilde{N}^*$ for the challenge ShillKey modulus and outputs as its guess the index of the modulus from the first phase that is closest to $\tilde{N}^*$.

Simulation results. We ran simulations of the above clustering attack for a mixed sample set of size $t \times k_1$ for various values of t (the number of cards) and values of k_1 equal to 100 and 1000. The results of our simulations are depicted in Figure 3.

Being a more ambitious type of attack, the success probabilities for this attack are considerably lower than for the previous attack. Nonetheless we see that our clustering approach is able to correctly identify a card with a probability of roughly 4 times the guessing probability. Notably, as we increase the values of k_1 and k_2 beyond 100 we do not get a corresponding increase in performance as in the previous attack. On the other hand, for low parameter values its performance is comparable to the previous attack. In fact if we compare Figures 2 and 3 we see that for $k_2 = 10$ the two attacks perform almost identically. Therefore if the attacker is limited to a small number of samples (≈ 10) during the identification phase, he can trace cards as effectively without requiring a sorted set of ciphertext samples during fingerprinting.

3.3 Connection to Key Privacy of RSA Encryption

We remark that our two ShillKey fingerprinting attacks only consider properties of RSA moduli and are, thus, of independent interest in the study of *key privacy* (or *key anonymity*) of RSA encryption, a security notion introduced by Bellare *et al.* in [BBDP01]. In the key privacy security model of [BBDP01], an adversary plays against two key pairs and is given both the public keys. Security is modelled in terms of key

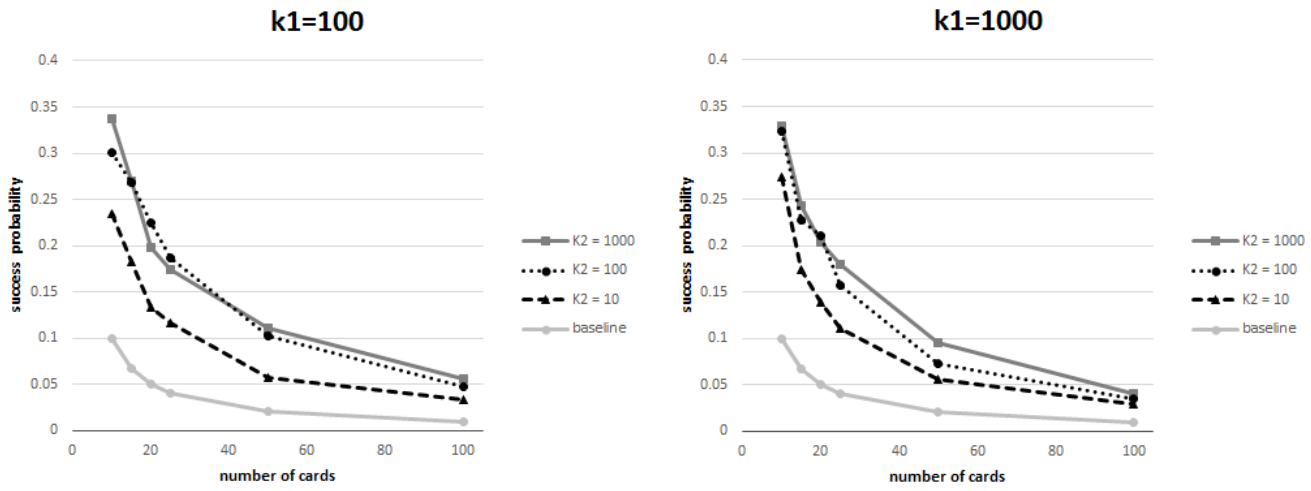

Figure 3: Simulations of the clustering attack for k_1 equal to 100 and 1000, and varying values of k_2 for each. The success probability is averaged over fifty runs, and the simulation was done with $t = 10, 15, 20, 25, 50, 100$ cards. The baseline indicates the success probability of an adversary that tries to win the game by pure guessing.

indistinguishability, requiring that it is infeasible for any efficient adversary, which can request encryptions of messages of its choice under one of the two public keys, to tell which key was chosen with probability higher than guessing. As already pointed out in [BBDP01], the RSA cryptosystem does *not* provide key privacy. Security is trivially broken when the two key lengths are different. However, RSA keys of the same bit length are easy to tell apart, too: let $N_0 < N_1$ be two RSA moduli: independently of the underlying plaintext, a ciphertext c computed under one of the two corresponding keys satisfies $c < N_b$. A single-query attack which succeeds with non-negligible advantage simply requests to encrypt an arbitrary message and then compares the resulting ciphertext c with the smallest modulus: if $c < N_0$ then it returns 0 and else guesses 1.

This attack is not directly applicable to the PLAID setting because there the RSA encryption keys are kept secret. Moreover, a realistic adversary against card untraceability should play against a number of cards $t \gg 2$. Still, our ShillKey fingerprinting attacks on PLAID can be seen as similar in spirit to, but obviously harder to perform than, the above single-query attack. Moreover, if the encryption scheme used in PLAID were to enjoy key privacy, then the attacks presented would be completely thwarted (and the RSA moduli would no longer need to be kept secret).

3.4 Countermeasures to Our Attacks

A very simple countermeasure to our attacks is for every card to use the same RSA ShillKey. This does not seem to have any negative security consequences and renders ineffective any tracing attacks based on the analysis of RSA ShillKey ciphertexts. A second countermeasure to our attacks is to modify the RSA encryption scheme so that it is key private. This can be done in two ways: padding by adding multiples of the modulus to the ciphertext, and selection of RSA moduli that all lie in a small interval.

4 Keyset Fingerprinting – Determining a Card's Capabilities

In this section we present another type of attack on PLAID's privacy, which we call *keyset fingerprinting*. This attack reveals the exact set of keys a card knows[8], thereby determining its *capabilities* in terms of which keyset it can use, i.e., which specific terminal it is able to talk to. In order to mount the attack, we exploit the following observations: (i) the KeySetIDs list sent by the terminal (in the clear) in the IA Command contains all keys known by the terminal [ISO14, 6.1], and is *not* authenticated[9], (ii) in its IA Response, the card is required to use the first key of the received KeySetIDs list it knows [ISO14, 6.2], and (iii) if the card uses its ShillKey, in the IA Response, then the terminal aborts [ISO14, 6.4].

4.1 The Attack in a Nutshell

We first explain the core idea of our attack by describing a concrete attack scenario. Assume an adversary observes a successful protocol run between a card and a terminal where the latter had sent (in the clear) KeySetIDs $= (2, 5, 8)$. From this, the attacker not only learns that the ICC holds at least one of the keys with IDs $\{2, 5, 8\}$, but it can also determine *all* of the keys the ICC supports, independently of the identifiers announced in KeySetIDs. To this end, the adversary can trigger a protocol run and mount a man-in-the-middle attack as described below.

In a first phase, the attacker sequentially replaces the IFD's original initial message by one containing only a single identifier from the original list of KeySetIDs, that is, 2, 5 or 8 in our example; by observing the subsequent protocol run, the attacker deduces that the ICC supports the selected key if and only if the protocol execution reaches the third step, i.e., if the terminal responds with a third message. In a second phase, the attacker sequentially prepends to the IFD's original initial message all key identifiers that were not contained in KeySetIDs, e.g., $(1, 2, 5, 8)$, $(3, 2, 5, 8)$, $\ldots$, $(65536, 2, 5, 8)$ in our example. Then, from each of the subsequent protocol runs, the attacker learns that the ICC knows the inserted key if and only if the IFD does *not* respond with a third message. This is because of observation (ii) above about the first matching key in the list to be used. At the end of the two phases, the attacker knows the identifiers of all keys supported by the ICC.

We stress the attack above can be performed in a remote fashion where two attackers, placed in physical proximity to the terminal (respectively, the card) relay the exchanged messages between each other, playing the role of a card (respectively, a terminal). Moreover, this attack can be mounted independently of the values announced in KeySetIDs, as long as the attacker observes a single, successful protocol execution.

Note that knowledge of all the keys supported by a card also reveals its capabilities (e.g., access authorizations), thereby potentially disclosing highly sensitive information. While this is not, in general, sufficient to identify a card uniquely, it effectively allows to derive capability classes, containing cards with the same capabilities. Moreover, in certain scenarios, capabilities like access authorizations might even leak the identity of a card's owner, hence breaking its anonymity, as some keys might be used exclusively to access security-critical infrastructure [ISO14, Annex C] such as, e.g., server rooms or the CEO's office. The impact of keyset fingerprinting is furthermore increased by the remote nature and the low cost of the attack (in terms of the number of interactions between terminal and card). Even in large-scale, realistic scenarios, the attack requires only few seconds (and no physical proximity of card and terminal) to determine a card's capabilities. See Section 4.2 for a more detailed discussion.

We remark that keyset fingerprinting can, in addition, be used as a prefilter for the tracing attacks

[8] Recall that terminals announce their supported keysets by sending corresponding KeySetIDs in the clear. As a consequence, any observer can see which keys are related to which resource/terminal.

[9] We note that the unauthenticated nature of the PLAID protocol messages has already been criticized in the national body comments on an earlier ISO draft [ISO13]. In our attack we exploit this weakness, refuting the claim of the current ISO draft [ISO14, Annex H.1.1] that sending KeySetIDs in clear is "of no use to an attacker."

discussed in Section 3 based on ShillKey Fingerprinting. Recall that the performance of these attacks heavily depends on the number of cards in the system that have to be distinguished. By first performing keyset fingerprinting on the card(s) in question, this number can potentially be reduced substantially (thereby improving the overall efficiency), as the ShillKey Fingerprinting in a second step only has to discriminate amongst the smaller number of cards belonging to the same capability class. Finally, we note that there are cases where the cheaper keyset fingerprinting attack on its own is actually already sufficient for a tracing attack: whenever a traced card has a *unique* set of supported keys (i.e., is the only member in its capability class), this attack is able to uniquely (re)identify that card. Furthermore, keyset fingerprinting suffices to distinguish two cards as long as there is a key supported by only one of the cards.

4.2 The Attack Details

Suppose that we observe a successful authentication between an honest terminal IFD and an honest card ICC. In the course of the protocol execution, the IFD starts by sending the list KeySetIDs = $(\text{KeySetID}_{i_1}, \ldots, \text{KeySetID}_{i_\ell})$ of (all) KeySetIDs it supports. The keyset fingerprinting attack proceeds in two phases, focusing first on the keys supported by the IFD and then on the remaining keys.

Phase 1. In the first phase, we replace the initial KeySetIDs list with a list containing only one of the keys supported by the IFD at a time, i.e., we replace the first message by (KeySetID_{i_j}) for $j = 1, \ldots, \ell$ in ℓ sequential interactions. We relay the response of the ICC unmodified to the IFD. If the IFD replies with a third message in the jth interaction, we can infer that the ICC knows the key with KeySetID_{i_j}. Otherwise, the ICC did not support this key and hence used its ShillKey, leading the IFD to abort.

Phase 2. In the second phase, we prepend the initial KeySetIDs list with one (or multiple, see below) values $\text{KeySetID}_j \notin \{\text{KeySetID}_{i_1}, \ldots, \text{KeySetID}_{i_\ell}\}$ at a time. We relay the response of the ICC unmodified to the IFD. If the IFD replies with a third message, we can infer that the ICC knows none of the prepended keys. Otherwise, the ICC did know at least one of these keys (which the IFD does not support), leading the IFD to abort. This relies on observations (ii) and (iii) above.

We measure the attack costs in terms of the number of interactions between IFD and ICC needed to extract the keys supported by the ICC. In the first phase, which requires ℓ interactions between the IFD and the ICC, we are able to determine exactly which of the keys $\{\text{KeySetID}_{i_1}, \ldots, \text{KeySetID}_{i_\ell}\}$ supported by the IFD the ICC knows. The second phase aims at determining which of the remaining $2^{16} - \ell$ KeySetIDs are known by the ICC. There are different strategies to proceed in Phase 2:

1. The *basic approach* is to simply prepend each one of the $2^{16} - \ell$ KeySetIDs not supported by the IFD one at a time, resulting in $2^{16} - \ell$ interactions in order to determine exactly which of the keys the ICC knows. Together with the first phase, this approach leads to 2^{16} interactions to fingerprint a card.

2. In the *binary search approach*, the set of KeySetIDs is partitioned along a binary tree with the full set of all $2^{16} - \ell$ KeySetIDs at the root, the first half of them as the left child, the second half of them as the right child, etc. In Phase 2, first the root (i.e., all $2^{16} - \ell$ KeySetIDs) is prepended. If the IFD replies, the ICC knows none of these keys and we have thus completed the keyset fingerprinting for the card. Otherwise, both the left half and the right half are prepended (sequentially) and, again, if the IFD replies then the ICC knows none of the prepended keys. This process can be repeated recursively until the IFD replies for each branch.

   Using this approach, we can quickly rule out those parts of the KeySetID space where the ICC does not know any key. More precisely, denote by n the number of keys the ICC knows in total and by ℓ' the number of keys the ICC knows amongst the ℓ keys supported by the IFD. Then we can upper bound the number of interactions needed to fingerprint a card by $(n - \ell') \cdot \log(2^{16}) + \ell = (n - \ell') \cdot 16 + \ell$,

since a traversal of the binary search tree in order to pinpoint a single key requires at most $\log(2^{16})$ (i.e., height of the tree) additional interactions.

3. The *binary search with known maximum approach* is a further optimization which is applicable in scenarios where the highest KeySetID in the system, MaxID, is known in advance. In this case, the binary tree can be reduced to the tree having only the MaxID $- \ell$ remaining unknown KeySetIDs (instead of all $2^{16} - \ell$) as leaves. The number of interactions to fingerprint a card therefore is reduced to $(n - \ell') \cdot \lceil \log(\text{MaxID}) \rceil + \ell$.

When comparing the strategies for the second phase, in the (unlikely) worst case where the card knows all 2^{16} possible keys, the basic approach requires 2^{16} interactions whereas the binary search approach takes approximately 2^{20} interactions. We observe however that the binary search is more efficient as long as the card holds less than 2^{12} keys (which we assume to be the case in any practical scenario).

A practical example. For the sake of providing the reader with some estimates on a more realistic, but still large-scale example, consider a scenario where MaxID $= 5000$ keys are deployed (enough for, e.g., a large building or a small campus) and the considered terminal and card both hold $\ell = n = 10$ keys, from which $\ell' = 1$ key is known by both.

We chose these parameters in the light of the targeted execution time for PLAID and the resource restrictions imposed by the terminal and card hardware. First, in every execution of the PLAID protocol, the terminal has to perform ℓ RSA decryptions, which is an expensive cryptographic operation for a computationally constrained embedded device.[10] But since the previous ISO draft aims at an overall protocol execution time of less than 300 ms [ISO13, p. 27], this means that ℓ cannot be too large. Second, the card has to store all of its n keys in a protected, tamper-proof memory [ISO14, Annex B]. As this kind of memory is very expensive, it is reasonable to assume that a card can store only a small number of keys. With these parameters, the binary search approach would require $(10 - 1) \cdot 16 + 10 = 154$ interactions to fingerprint the card which, knowing the highest KeySetID MaxID, can be optimized to $(10 - 1) \cdot \log(5000) + 10 \approx 121$ interactions using the binary search with known maximum.

4.3 Potential Countermeasures Against Our Attack

As the keyset fingerprinting attack relies heavily on the malleability of the initial KeySetIDs message sent by the terminal, tamper-protecting this message is the obvious way to prevent this attack. One potential and immediate remedy to detect and to prevent tampering with the initial message would be to let the ICC include a hash value of the KeySetIDs value in the plaintext of STR1. The terminal could then check whether the ICC obtained the unmodified initial message by comparing the hash value that it receives with the hash of the original KeySetIDs value. However, a rigorous analysis would be required to put this idea on a profound foundation.

5 Further Security Considerations

Here we discuss further security considerations, and mainly the secrecy of the established keys which, according to the standard, can be optionally used "as a secure messaging, session or encryption key in subsequent sessions." We also point out that the design of PLAID also deviates in several ways from good cryptographic practice. We observe that some of these issues have already been pointed out in the comments on the previous ISO draft version [ISO13].

[10] For 2048-bit RSA decryptions or signatures, [RPHJ11] reports times of over 100 ms for mobile devices, while our simulations on an Intel Core i7 2.4 GHz are around 10 ms.

5.1 Forward (In)security

Forward security [BPR00] demands that one cannot recover session keys generated in the past, even if the long-term secrets of a party become known. In the case of PLAID, the long-term secrets correspond to the secret RSA keys and the FAKey on the terminal side, and to the public RSA keys, DivData, and the diversified keys $\text{FAKey}^{(\text{Div})}$ on the card side. The loss of keys of either party immediately reveals all past session keys, and also of future sessions, even if they are executed honestly between the parties and the adversary merely observes these execution traces. Furthermore, revealing a card's secrets also allows the identification, *a-posteriori*, of traces belonging to that card and so breaches privacy in this sense.

Assume first that a terminal's long-term secrets become known to the adversary, and consider the trace of an execution between this terminal with an arbitrary card: the adversary can, analogously to the genuine server, try to decrypt the ciphertext encrypting string ${}^{e}\text{STR1}$ under all possible RSA private keys of the terminal, until it succeeds with one key. It then obtains DivData, hence can compute $\text{FAKey}^{(\text{Div})}$ by executing $\text{AES}^{\text{FAKey}}_{\text{Encrypt}}(\text{DivData})$ and then decrypt ${}^{e}\text{STR2}$ sent by the honest terminal to recover the session key KeysHash.

Next, suppose that the adversary gets hold of the diversification data DivData and the diversified key $\text{FAKey}^{(\text{Div})}$ of a card. It can then try to decrypt ${}^{e}\text{STR2}$ with this key to obtain some candidate KeysHash for the session key. The adversary can verify the validity of this candidate by checking that ${}^{e}\text{STR3}$ decrypts under the candidate key to the given DivData. This way, the adversary is able to identify traces belonging to the specific card and to determine correct session keys of the card.

Most importantly, any such breach would lead to the disclosure of the payload data which may be highly sensitive (for example, a user's biometric data).

5.2 Key (In)security in the Bellare–Rogaway Model

The PLAID protocol specifies the option of reusing the negotiated session key KeysHash for subsequent secure communication. We comment on possible consequences of doing so. Our starting point is the widely-used Bellare–Rogaway (BR) security model [BR93] for key exchange protocols. This model demands that all session keys should look random to the adversary. Neglecting technical details, this is formalized by presenting the adversary either the genuine session key or an independent random key and challenging it to decide which is the case. This immediately requires of a protocol that its session keys are not themselves used in a non-trivial way in the key exchange steps, otherwise the adversary can try to test the given key against a protocol execution trace. In the specific case of PLAID, the adversary can try to decrypt ${}^{e}\text{STR3}$ with the given key, and will recover a meaningful plaintext with overwhelming probability if and only if this key equals the genuine key KeysHash. Thus PLAID *cannot* achieve security in the BR model.

Note that the lack of security in the BR sense does not necessarily imply that a protocol is insecure. It merely means that other models must be used to assess its security. PLAID is not unique in this respect: a prominent example of a protocol not achieving BR security is TLS, leading researchers to investigate various alternative security evaluations [JKSS12, KPW13, BFS+13, GKS13, BFK+14]. The usage of the session key in the exchange step is often alleviated by the fact that messages in this part and in the channel protocol differ in format, e.g., if a counter value is used and incremented with each application. This form of "domain-separation", however, is not necessarily given in case of PLAID, because the subsequent channel message format has not been specified.

Interestingly, PLAID could easily avoid the problems with the session key being used in the key exchange phase. Recall that the session key KeysHash for AES (with 128 bits) is derived as the first 128 bits of the hash value $\text{SHA-256}(\text{RND1} \,\|\, \text{RND2})$. Since the hash value has 256 bits one could easily use the remaining 128 bits as the AES-128 key for the final message in the key exchange step, and then switch to KeysHash as before in the channel protocol. In the original protocol the card in some sense demonstrates knowledge

of FAKey$^{(\mathrm{Div})}$ by being able to decrypt the terminal's message and answer under the derived key. This would still be true with the proposed modification. Note however that this modification still requires a formal security treatment.

5.3 On the Applicability of Bleichenbacher's Attack

Recall that PLAID uses PKCS#1 v1.5 padding for RSA encryption. The accompanying protocol description [Cen09] argues that there is no need to use OAEP padding, because "PLAID doesn't expose the modulus or any other RSA primitive" and that "there is a significant performance advantage in using PKCS#1 v1.5 padding." While we do not feel inclined to comment on the performance related issue, the first part of the argument is debatable in light of the fact that exposure of a card's secrets does reveal the public keys. Further, our attacks in the previous sections show that some information about the moduli is revealed, and the exponent e may be fixed. We note that the comments section in the previous ISO version of PLAID [ISO13] also asks for investigations of the possibility of mounting Bleichenbacher's attack.

Once the RSA public key is known one can in principle mount Bleichenbacher's attack [Ble98] on PKCS# v1.5 padding. In this attack the adversary takes a ciphertext $c \in \mathbb{Z}_N^*$ of some unknown padded message m and "shifts" the message by multiplying c with a random $s^e \bmod N$. With sufficiently high probability the derived "message" $sm \bmod N$ is PKCS#1 v1.5 padding compliant. The adversary could thus potentially deduce information about m in case of an error message[11] indicating correct or incorrect padding, and given sufficiently many error messages, recover m. The attack has been significantly improved in a series of papers, e.g., [JSS12, MSWS14].

For PLAID, the message format carries some redundancy in terms of repeating RND1. Hence, most likely the shifted message $sm \bmod N$ will not be accepted by the terminal in any case, independently of the padding. However, the detailed behaviour is implementation-specific. For example, the current implementation is based on the JavaCard framework and the decryption procedure of PKCS#1 v1.5 merely throws an exception in case of incorrect padding and leaves it up to the higher level program to treat this exception.

5.4 CBC-mode Encryption

PLAID proposes to use CBC-mode encryption based on AES. The standard explicitly demands that the initialization vector IV is set to the all-zero string for both ^{e}STR2 (from the terminal to the card) and ^{e}STR3 (from the card to the terminal). This usage does not conform with standard practice, which demands the use of random IVs to achieve security against chosen plaintext attacks. As remarked before, the PLAID specification states that padding is only applied "if necessary" and is thus not compliant with ISO/IEC 9797-1 padding method 2, where padding is always applied. Indeed, this imprecision makes the standard unimplementable as currently specified, since there will be cases arising during decryption where it is not possible to discern whether padding should be removed or not. It is well-known that CBC-mode encryption is especially vulnerable to padding oracle attacks [Vau02], and that careful implementation is needed to avoid them. The lack of precision in this aspect of the PLAID specification does not bode well.

It is also now well-understood in the cryptographic community that CBC-mode encryption does not offer sufficient integrity guarantees on its own to provide adequate security against active attacks. The usual solution is to add explicit integrity protection through the application of a MAC algorithm to the CBC-mode ciphertext. PLAID does not do so, and a justification for why this lack of integrity does not endanger security was requested in the comments of the previous ISO standard draft [ISO13], but was not addressed in the latest version [ISO14].

[11] The protocol explicitly notes that no error messages should be issued, but wrong implementations or side-channel attacks may reveal such information.

PLAID does offer mild forms of plaintext integrity. For example, STR2 contains the session key KeysHash computed as the hash of RND1 $\|$ RND2 while STR3 contains DivData. These elements can be checked for after decryption by the relevant party, and this would detect some forms of adversarial plaintext manipulation through simple bit-flipping in the corresponding ciphertexts ^{e}STR2 and ^{e}STR3. However, it is easy to see that an attacker can still manipulate other fields in STR2 and STR3 by bit-flipping in ciphertexts (even with a fixed IV). While this lack of integrity has not led us to the discovery of specific attacks on PLAID, it is a worrying feature that could be easily avoided through the application of mainstream cryptographic design principles.

5.5 Entity Authentication

Note that both parties, IFD (reader) and ICC (card), basically authenticate one another by proving knowledge of a secret key. For the terminal this is done via the secret RSA key, whereas the card uses its unique DivData and therefore unique key $\text{FAKey}^{(\text{Div})} = \text{AES}^{\text{FAKey}}_{\text{Encrypt}}(\text{DivData})$. The standard mechanism to do so would be either to compute a signature or a message authentication code for a random challenge, or to return, in clear, a nonce encrypted under the party's key.

PLAID follows the encryption-based approach. Yet the usual security argument for this type of authentication requires chosen-ciphertext security for the deployed encryption scheme. PLAID, on the other hand, uses two encryption schemes which are known not to provide this level of security, i.e., RSA-PKCS#1 v1.5 and plain AES-CBC-encryption. This does not mean that the protocol is insecure and does not provide any form of entity authentication. However one cannot infer security from known results but would instead need carefully constructed *de novo* arguments.

5.6 Payload Insecurity

During the ISO standardization process the PLAID protocol was changed to introduce an optional payload field in the third protocol message, the second message from the IFD to the card (see Step 5 in Figure 1) [ISO13]. The standard motivates the purpose of this payload field—this field should not be confused with the payload field in the last message by the card in Step 7 (Figure 1)—for scenarios where, for example, a user enters a PIN and the verification should be done on the card. In this case, the PIN is to be sent to the card within the payload field [ISO14, Annex G]. The problem is that sensitive information send by the IFD can always be intercepted via a simple man-in-the-middle attack, assuming an adversary has corrupted any card. Breaking a card allows the attacker to learn the diversified $\text{FAKey}^{(\text{Div})}$ of the card, the card's DivData field as well as the public part of one (or more) IAKeys. Thus, an adversary can simply replace the second message during an honest execution with one corresponding to the broken card. This will lead to the IFD encrypting the third message under the $\text{FAKey}^{(\text{Div})}$ of the broken card and hence, if a PIN (or any other payload) is included, the adversary can trivially learn it by a simple decryption operation. Note that this problem is not due to the payload being sent in the third message, but that a user, when entering a PIN, cannot tell whether or not the terminal is actually communicating with his/her card. Therefore, PIN comparison on the card as proposed [ISO14, Annex G] is generically insecure due to the given attack scenario.

5.7 On the Impossibility of Key Revocation

Although PLAID uses a public-key encryption system (i.e. RSA) during the initial authentication phase, the overall setup resembles more a symmetric setting where all static keys used by parties are exchanged during system setup (abstracting away the diversification procedure of PLAID). As a consequence, it is not possible to revoke any compromised keys within PLAID. In order to exemplify the resulting consequences, assume that an attacker is able to break into an IFD (terminal). The IFD contains a list of IAKeys and a

list of FAKeys which thereby are revealed to the attacker. With this information, the attacker can generate arbitrary new cards with the capabilities of any of the KeySetIDs known by the broken IFD. Furthermore, there is no way to revoke the compromised keys in the system without issuing new cards, as the keys known by IFDs are hardcoded into the cards. Thus, even the break of a single IFD can lead to an entire PLAID setup becoming insecure.

5.8 Key Legacy Attack

A *key legacy attack* is related to the same issues allowing for the KeySet fingerprinting attack, namely, the lack of authentication in the list of keys used by a card and a terminal to establish a connection. Recall that the protocol specifies that the first commonly shared key in the list has to be used, even if there are other shared keys. This means that an adversary could force the card to use one particular key (among those supported by both card and terminal) by *reordering* the list of keys sent by the terminal in a man-in-the-middle fashion. This could be dangerous in case one or more of the keys in the system are compromised, or turn out to provide inferior security for any reason, even if the use of these keys is de-prioritized (e.g., by having the terminals set them always last in order of preference). We note that this type of attack was already mentioned in the national body comments to the first ISO draft [ISO13], but remained unconsidered in the current version [ISO14].

6 Conclusion

Our results show that PLAID has significant privacy weaknesses. The shill key attack and the keyset fingerprinting attack reveal card identifying information and, via access authorizations, information about the card holder. As for entity authentication and the secrecy of established keys for subsequent communication, in several places the design of PLAID follows some uncommon strategies and reveals potential attack vectors, such as the lack of forward security. The case of PLAID also shows that standards should specify details thoroughly, in order to avoid vulnerable implementations. An example here is the ISO/IEC 9797-1 non-compliant CBC padding in PLAID, which potentially enables padding attacks (see our remark in Section 5).

We do not recommend the indiscriminate usage of PLAID in its current form, especially not for privacy-critical scenarios. While our proposed countermeasures seem to thwart our attacks on privacy, a more comprehensive analysis of the protocol in light of clearly stated security goals would be necessary. The PLAID description promises that the protocol should be scrutinized by "the most respected cryptographic organisations, as well as the broader cryptographic community" [Cen09]. Unfortunately, we are not aware of any available documents in this regard. Indeed, standardization processes in general would benefit if supporting material, arguing the security of a proposal, was available at the time of evaluation.

Acknowledgments

We thank Pooya Farshim for his contributions during the early stages of this paper and the anonymous reviewers for valuable comments. Marc Fischlin is supported by the Heisenberg grants Fi 940/3-1 and Fi 940/3-2 of the German Research Foundation (DFG). Tommaso Gagliardoni and Felix Günther are supported by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE. Felix Günther and Giorgia Azzurra Marson are supported by the DFG as part of the CRC 1119 CROSSING. Giorgia Azzurra Marson and Arno Mittelbach are supported by the Hessian LOEWE excellence initiative within CASED. Kenneth G. Paterson and Jean Paul Degabriele are supported by EPSRC Leadership Fellowship EP/H005455/1.

References

[Aus10] Standards Australia. *AS 5185-2010 Protocol for Lightweight Authentication of IDentity (PLAID)*. Standards Australia, 2010. (Cited on pages 3 and 4.)

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 566–582, Gold Coast, Australia, December 9–13, 2001. Springer, Berlin, Germany. (Cited on pages 11 and 12.)

[BFK+14] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella Béguelin. Proving the TLS handshake secure (as it is). In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 235–255, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Berlin, Germany. (Cited on page 16.)

[BFS+13] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, 12(4):267–297, 2013. (Cited on page 16.)

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Berlin, Germany. (Cited on page 17.)

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Berlin, Germany. (Cited on page 16.)

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1993. Springer, Berlin, Germany. (Cited on page 16.)

[Cen09] Centrelink. Protocol for Lightweight Authentication of Identity (PLAID) — Logical Smartcard Implementation Specification PLAID Version 8.0 - Final, December 2009. (Cited on pages 3, 4, 5, 17, and 19.)

[Dep14] Department of Human Services. Protocol for Lightweight Authentication of Identity (PLAID), 2014. (Cited on page 3.)

[DFG+13] Özgür Dagdelen, Marc Fischlin, Tommaso Gagliardoni, Giorgia Azzurra Marson, Arno Mittelbach, and Cristina Onete. A cryptographic analysis of OPACITY - (extended abstract). In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 345–362, Egham, UK, September 9–13, 2013. Springer, Berlin, Germany. (Cited on page 5.)

[Fre14] Graeme Freedman. Personal communication by e-mail, July 2014. (Cited on page 5.)

[GKS13] Florian Giesen, Florian Kohlar, and Douglas Stebila. On the security of TLS renegotiation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 387–398, Berlin, Germany, November 4–8, 2013. ACM Press. (Cited on page 16.)

[ISO13] ISO. *DRAFT INTERNATIONAL STANDARD ISO/IEC DIS 25185-1 Identification cards — Integrated circuit card authentication protocols — Part 1: Protocol for Lightweight Authentication of Identity*. International Organization for Standardization, Geneva, Switzerland, 2013. (Cited on pages 4, 5, 13, 15, 17, 18, and 19.)

[ISO14] ISO. *DRAFT INTERNATIONAL STANDARD ISO/IEC DIS 25185-1.2 Identification cards — Integrated circuit card authentication protocols — Part 1: Protocol for Lightweight Authentication of Identity*. International Organization for Standardization, Geneva, Switzerland, 2014. (Cited on pages 3, 4, 5, 6, 9, 13, 15, 17, 18, and 19.)

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Germany. (Cited on page 16.)

[Joh94] Roger Johnson. Estimating the size of a population. *Teaching Statistics*, 16(2):50–52, 1994. (Cited on page 9.)

[JSS12] Tibor Jager, Sebastian Schinzel, and Juraj Somorovsky. Bleichenbacher's attack strikes again: Breaking PKCS#1 v1.5 in XML encryption. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012: 17th European Symposium on Research in Computer Security*, volume 7459 of *Lecture Notes in Computer Science*, pages 752–769, Pisa, Italy, September 10–12, 2012. Springer, Berlin, Germany. (Cited on page 17.)

[Kli10] Ryan Kline. Improving contactless security is goal of emerging PLAID project. `http://secureidnews.com/news-item/improving-contactless-security-is-goal-of-emerging-plaid-project/`, January 2010. SecureIDNews. (Cited on page 3.)

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany. (Cited on page 16.)

[KR12] Koh Ho Kiat and Lee Yong Run. An analysis of OPACITY and PLAID protocols for contactless smart cards. Master's thesis, Naval Postgraduate School, Monterey, CA, USA, September 2012. (Cited on page 4.)

[MSWS14] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, and Joerg Schwenk. Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks. In *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, August 2014. USENIX Association. (Cited on page 17.)

[Nat09] National Institute of Standards and Technology. Protocol for Lightweight Authentication of Identity (PLAID) Workshop, July 2009. (Cited on page 3.)

[Ris09] Risky.biz. Risky Business 106 — Centrelink's new PLAID auth protocol. http://risky.biz/netcasts/risky-business/risky-business-106-centrelinks-new-plaid-auth-protocol, May 2009. (Cited on page 5.)

[RPHJ11] Helena Rifà-Pous and Jordi Herrera-Joancomartí. Computational and energy costs of cryptographic algorithms on handheld devices. *Future Internet*, 3(1):31–48, 2011. (Cited on page 15.)

[Sak13] Hideki Sakurada. Security evaluation of the PLAID protocol using the ProVerif tool. http://crypto-protocol.nict.go.jp/data/eng/ISOIEC_Protocols/25185-1/25185-1_ProVerif.pdf, September 2013. (Cited on page 4.)

[Tay12] Josh Taylor. Centrelink ID protocol still in trial phase. http://www.zdnet.com/centrelink-id-protocol-still-in-trial-phase-1339336953/, May 2012. ZDNet. (Cited on page 3.)

[Vau02] Serge Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ... In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany. (Cited on page 17.)

[Wat13] Dai Watanabe. Security analysis of PLAID. http://crypto-protocol.nict.go.jp/data/eng/ISOIEC_Protocols/25185-1/25185-1_Scyther.pdf, September 2013. (Cited on page 4.)