On the Joint Security of Encryption and Signature in EMV[⋆]

Jean Paul Degabriele[1], Anja Lehmann[2], Kenneth G. Paterson[1],
Nigel P. Smart[3], and Mario Strefler[4]

[1] Information Security Group, Royal Holloway, University of London
[2] IBM Research – Zurich
[3] Department of Computer Science, University of Bristol
[4] INRIA / ENS / CNRS, Paris

Abstract. We provide an analysis of current and future algorithms for signature and encryption in the EMV standards in the case where a single key-pair is used for both signature and encryption. We give a theoretical attack for EMV's current RSA-based algorithms, showing how access to a partial decryption oracle can be used to forge a signature on a freely chosen message. We show how the attack might be integrated into EMV's CDA protocol flow, enabling an attacker with a wedge device to complete an offline transaction without knowing the cardholder's PIN. Finally, the elliptic curve signature and encryption algorithms that are likely to be adopted in a forthcoming version of the EMV standards are analyzed in the single key-pair setting, and shown to be secure.

1 Introduction

According to the EMV Co website[1],

EMV is a global standard for credit and debit payment cards based on chip card technology. As of end-2010, there were more than 1.24 billion EMV compliant chip-based payment cards in use worldwide. EMV chip-based payment cards, also known as smart cards, contain an embedded microprocessor, a type of small computer. The microprocessor chip contains the information needed to use the card for payment, and is protected by various security features.

The EMV standards [14–17] are a complex set of documents defining all aspects of the system, many unrelated to cryptography. In particular, they define three main protocols, SDA, DDA and CDA, which offer different levels of card (and card-holder) authentication and transaction authorization. A good overview of the EMV system can be found in [26], for example.

This paper concerns the security of the encryption and signature schemes used in the EMV standards. These are currently based on the RSA primitive, but there are plans to move to elliptic curve based cryptography at some future date, with a draft specification for ECC in EMV having been available for several years [13] and specifying the use of EC-DSA for signatures and ECIES for encryption, and a more recent announcement[2] that EC-Schnorr is under consideration. A main motivation for switching to ECC is that the EMV standards have data formats that do not allow public keys to be larger than 1984 bits.

EMV makes use of signatures during card authentication in DDA and for transaction authentication in CDA. The latter gives the terminal an opportunity to verify the transaction's integrity, without needing to go on-line to communicate with the back-end banking infrastructure to obtain such an assurance. As an option, EMV allows public key encryption to be used for PIN encryption, encrypting the PIN entered by the card-holder at a Point-of-Sale (PoS) terminal to protect it as it is transferred from the terminal to the card, where it is decrypted and compared to the PIN stored on the card. In addition, EMV makes extensive use of symmetric key techniques, but these will not concern us here.

[⋆] An abridged version of this work appears at CT-RSA 2012. This is the full version.
[1] `http://www.emvco.com/`
[2] See `http://www.emvco.com/faq.aspx?id=38`.

The EMV standards allow the same RSA key-pair to be used for both signature and encryption (see [15, Section 7.1]). This brings benefits in terms of reducing the number of certificates needed in the system, which in turn reduces their storage and processing costs. Our understanding is that the same will be permitted in future versions of EMV using ECC. But is this practice secure? This question is what our paper sets out to address.

Joint security of signature and encryption: The topic of *joint security* of signature and encryption schemes has a fairly extensive history. The first paper to study the problem formally seems to have been by Haber and Pinkas [19], who introduced the concept of a *combined public key scheme* where the existing encrypt, decrypt, sign and verify algorithms of a signature and an encryption scheme are preserved, but where the two key generation algorithms are modified to produce a single algorithm. This algorithm still outputs two key-pairs, but the key-pairs are no longer necessarily independent, and may even be identical. Haber and Pinkas also introduced the natural security model for combined public key schemes, where the adversary against the encryption part of the scheme is equipped with a signature oracle in addition to the usual decryption oracle, and where the adversary against the signature part of the scheme is given a decryption oracle in addition to the usual signature oracle. In this setting, we talk about the *joint security* of the combined scheme. Further work on this topic can be found in [6, 25] where the special case of combined schemes built from trapdoor permutations (including the RSA trapdoor permutation) was considered. A recent paper [29] revisits this topic, giving constructions that are provably secure in the standard model and a pathological example showing that schemes that are individually secure may become catastrophically insecure when the same key-pair is used for signature and encryption. That paper also notes that textbook RSA is trivially insecure in such a setting, since access to a decryption oracle (which on input c outputs $m = c^d \bmod N$) instantly allows signature forgeries, simply by setting $c = H(m)$. Fortunately, EMV does not use textbook RSA, so this attack does not apply.

Previous work on EMV security: Coron *et al.* [8] give an existential forgery attack on the ISO/IEC 9796-2 signature standard, which is also used by EMV. They improved on a previous attack by Coron *et al.* [23], itself an extension of an attack by Desmedt and Odlyzko [11] to larger messages. For operational reasons, these attacks do not threaten the security of EMV signatures. Coron *et al.* [7] showed how to factor the RSA modulus using fault attacks on signatures found in the EMV standard. Using about ten signatures, their attack recovers the factors of N in less than a second. More recently, Smart [33] used a ciphertext validity checking oracle that may exist in certain implementations of the EMV standard to recover the PIN from its encrypted version using as few as 30 queries to the oracle.

At the protocol level, Murdoch *et al.* [26] reported an implementation of a “wedge attack” on the EMV protocol. In the attack, the communications between a PoS terminal and a card are interfered with using a wedge device. This device prevents the PIN validation request from reaching the card and returns a “PIN valid” code to the terminal. This leaves the card and terminal with different views of the protocol, but the card’s view of the protocol may not be transmitted to the terminal in a form that is interpretable by the terminal, so the modification may go undetected. Murdoch *et al.* explain how this attack could enable an attacker to make use of a stolen card while not knowing the cardholder’s PIN. Their attack may work for all three protocol versions SDA, DDA and CDA. It is notable that this class of attack was already anticipated well before the publication of [26], with protections for CDA already existing in, for example, the EMV Common Payment Application (CPA) specification [12].

Our contribution: None of the previous analysis of EMV examines the security consequences of using the same key-pairs (whether RSA- or ECC-based) for both signature and encryption. Our paper does so, with results that are both negative and positive for EMV:

– We present a theoretical attack against EMV’s existing RSA-based algorithms which shows how adversarial access to error information that may be generated during decryption can be used to forge a signature on a freely chosen message. We then show how the attack can be integrated into EMV’s CDA protocol flow, enabling an attacker with a wedge device to complete an offline transaction without knowing the cardholder’s PIN. Note that this attack would still work even if the proposed countermeasures to the attacks of [26] were adopted. However, the attack is unlikely to work in practice because of other factors which we describe in detail in Section 3.

– We provide positive security results in the joint security setting for the ECC-based encryption and signature schemes that are proposed in [13] and that are likely to be adopted in future EMV standards. More specifically, we prove that the ECIES encryption scheme and the EC-Schnorr signature scheme are jointly secure in the Random Oracle Model, and that ECIES and the EC-DSA signature scheme are jointly secure in the Generic Group Model (GGM). Such results are the best that we can currently hope for, given the state of the art in analyzing ECC signature schemes.

Paper organisation: In the next section, we present our RSA signature forgery attack. In Section 3 we explain how this attack might be used in the context of the EMV protocols to forge a transaction signature for the CDA protocol. Section 4 presents our analysis of the joint security of EMV's likely choices for ECC-based algorithms. We conclude in Section 5.

2 An Attack on Combined Signature and Encryption Schemes from EMV

In this section, we show an attack on the combined signature and encryption scheme from EMV in which an adversary, equipped with a partial decryption oracle, is able to forge a signature on a message of his choice. The partial decryption oracle only tells the adversary whether or not the underlying plaintext is correctly formatted, and the attack is therefore closely related to the attack of Bleichenbacher [3] on RSA with PKCS#1 encoding. However the low-level details differ because of the specific format used in EMV encryption. Note that the attack works independent of the particular encoding used when creating signatures. The idea of using a (partial) decryption oracle to forge signatures in the RSA setting was mentioned in passing in [3] and examined in more detail in [24]. In the next section, we examine the applicability of this attack in the context of the EMV protocol.

2.1 Description of the attack

According to [15], Table 25, the plaintext to be encrypted using the ICC's RSA public key (e, N) consists of a 1-byte Data header equal to `7F`, followed by an 8-byte PIN block encoding the PIN in a particular format, followed by the 8-byte ICC Unpredictable Number, and then a final $k - 17$ bytes of random padding. Here k is the length of the ICC's RSA public key in bytes, and is referred to as N_{IC} in the EMV standards.

In what follows, we assume that the decryption process fails with an output of invalid if the obtained plaintext does not begin with a byte `7F`, and otherwise outputs valid. The justification for making this assumption in an EMV context is given in more detail in Section 3. To model this, we equip the adversary with an oracle valid$(\cdot)$ that returns either `true` or `false` on input a ciphertext c.

Let k be the byte length of N (so $2^{8(k-1)} \leq N < 2^{8k}$). The 4-digit PIN is first encoded as an 8-byte PIN block $P = \mathtt{24}||pp||pp||(\mathtt{FF})^5$, where $pp||pp$ denotes a BCD encoding of the digits of the PIN. The PIN block is in turn encoded as a plaintext $m = \mathtt{7F}||P||U||R$, where U is the 8-byte ICC Unpredictable Number and R is the $k - 17$ bytes of random padding. A ciphertext c is valid if its decryption m starts with `7F`, so for $B = 2^{8(k-1)}$ we have $127B \leq m < 128B$ when c is valid. We write L for the lower bound $127B$, U for the upper bound $128B - 1$. The modulus N has to be larger than this, so $128B \leq N < 256B$.

Let m be the message for which we wish to forge a signature σ, and let $\mu(m)$ denote the encoding that is applied to m before the signing operation, so that $\sigma = \mu(m)^d \bmod N$. For our attack, μ is arbitrary, but of course EMV uses a specific encoding function. Our attack is then presented in Algorithm 1. The attack is divided into three steps: blinding, a search for valid ciphertexts, and narrowing down the set of solutions. The blinding step is executed once at the start, and is then followed by a series of iterations where each iteration comprises searching for a new valid ciphertext (step 2) and updating the set of solutions accordingly (step 3). The search is itself divided into three cases of which in each iteration only one will be executed. Step 2a is executed only on the first iteration. If it results in more than one interval, step 2b will be executed in order to reduce this set of intervals to just one. Bleichenbacher [3] presents a heuristic argument to show that a

key length	prediction	max.	95th p.	90th p.	median	10th p.	5th p.	min.
512	1408 – 1792	1149495	5764	2837	1462	1166	1088	836
768	1920 – 2304	344278	5900	3318	2029	1682	1594	1233
1024	2432 – 2816	868159	10647	4660	2577	2106	2032	1610
1984	4320 – 4704	221440	17385	8524	4639	3855	3650	3216

Table 1. Number of queries needed to attack different key lengths

single iteration of step 2b will in most cases be enough. Once the set of solutions is reduced to one interval, each iteration of step 2c will attempt to halve this interval until it is narrowed down to a single value.

Complexity We estimate the number of oracle accesses in a way analogous to Bleichenbacher [3]. The probability $\Pr(A)$ that a randomly chosen integer $0 \leq m < N$ begins with the byte `7F` is bounded by

$$2^{-8} < \Pr(A) < 2^{-7}$$

if we assume that the bitlength of the modulus N is a multiple of 8 (as it is in EMV). Because we assume that the decryption oracle first checks whether the recovered plaintext starts with `7F` and that we can learn the result of this test, the probability that our ciphertext passes this test is $\Pr(P) = \Pr(A)$. Therefore, step 1 needs about $1/\Pr(P) \in [2^7, 2^8]$ decryption queries on average. Step 2a will take the same number of queries on average, as does each execution of step 2b. Step 2c should on average take 2 queries, since we are trying to reduce the interval by half each time. If we assume that step 2b is executed once, and the modulus has $\log N$ bits, this yields an expected number of queries in the range

$$[3 \cdot 2^7 + 2 \cdot \log N, 3 \cdot 2^8 + 2 \cdot \log N].$$

2.2 Experimental Results

In order to experimentally validate our theoretical predictions of the number of oracle queries needed to forge a signature, we implemented Algorithm 1 in maple and ran it 1000 times for each of four different key lengths, 512 bits, 768 bits, 1024 bits and 1984 bits. The encoded message $\mu(m)$ was picked uniformly at random from the interval $[0, N-1]$ for each trial. While this choice does not respect the EMV encoding, the message is always blinded in step 1, so results are not affected. Figures 3 to 6 in Appendix A depict the distribution of the number of runs that needed roughly x queries versus x for each keylength. We cut off the top 5% values in each data set in order to better display the distributions. Table 1 gives an overview of the results.

The first thing to notice is the length of the distribution's tail, as evidenced by the large difference between the 95th percentile and the maximum in each row of Table 1. Similar behaviour was hinted at by Bleichenbacher [3] who commented "The probabilities in this section were computed under the assumption that the values s_i are independent of each other. [. . .] However, the assumption may be wrong in special cases.[. . .] in certain situations the attack might require many more chosen ciphertexts than our analysis indicates.". On the positive side, we note that the median lies nicely within the expected range, and the 90th and 95th percentile are below twice, respectively four times, the median. This means that the algorithm behaves well in the majority of cases.

3 Application of the Attack to EMV

In this section, we study to what extent the above attack can be realized in the context of the EMV protocols, and what the impact of the attack would be. We begin by studying in more detail how PIN decryption is specified in EMV, and then examine how the forgery attack can be realized in the context of an offline CDA transaction.

```
$c \leftarrow \mu(m)$, $s_0 \leftarrow 1$;
while $\neg\text{valid}(c{s_0}^e \bmod N)$ do $s_0 \xleftarrow{\$} \mathbb{Z}_N$ ;                // step 1
$c \leftarrow c{s_0}^e \bmod N$;
$M \leftarrow \{[L, U]\}$;
$i \leftarrow 1$;
while $M \neq \{[a, a]\}$ do
    if $i = 1$ then                                                    // step 2a
        $s \leftarrow \lceil \frac{N+L}{U} \rceil$;
        while $\neg\text{valid}(cs^e \bmod N)$ do $s \leftarrow s + 1$;
    end
    if $i > 1 \wedge |M| > 1$ then                                     // step 2b
        $s \leftarrow s + 1$;
        while $\neg\text{valid}(cs^e \bmod N)$ do $s \leftarrow s + 1$;
    end
    if $i > 1 \wedge |M| = 1$ then                                     // step 2c
        $\{[a, b]\} \leftarrow M$, $s_{\text{prev}} \leftarrow s$;
        $r \leftarrow \lceil 2\frac{bs_{\text{prev}} - L}{N} \rceil$;
        flag $\leftarrow$ false;
        while $\neg$flag do
            if $\lfloor \frac{L+rN}{b} \rfloor < \lfloor \frac{U+rN}{a} \rfloor$ then
                $s \leftarrow \lceil \frac{L+rN}{b} \rceil$;
                flag $\leftarrow \text{valid}(cs^e \bmod N)$;
            end
            $r \leftarrow r + 1$;
        end
    end
    $I \leftarrow \emptyset$;
    foreach $[a, b] \in M$ do                                          // step 3
        for $r \leftarrow \lceil \frac{as-U}{N} \rceil$ to $\lfloor \frac{bs-L}{N} \rfloor$ do
            $I \leftarrow I \cup \{[\max(a, \lceil \frac{L+rN}{s} \rceil), \min(b, \lfloor \frac{U+rN}{s} \rfloor)]\}$
        end
    end
    $M \leftarrow I$, $i \leftarrow i + 1$;
end
return $\sigma \leftarrow as_0^{-1} \bmod N$
```

Algorithm 1: Forging Algorithm

Decryption processing: From [15, Section 7.2] and [16, Section 10.5.1], the ICC (integrated circuit card, or chip) carries out a particular sequence of steps when decrypting. In part, the steps are as follows (we preserve numbering from [15]):

6. Use the ICC private key to decrypt the enciphered PIN data.
7. Check that the ICC Unpredictable Number recovered is equal to the ICC Unpredictable Number that was generated by the ICC with the `GET CHALLENGE` command.
8. Check whether the Data Header byte recovered is equal to 7F.
9. Check whether the PIN in the recovered PIN block corresponds with the PIN stored in the ICC.

Each of steps 6-9 above may fail, in which case the ICC returns an error code `6983` or `6984`. There is one exception, which arises as part of step 9. Here, if all the format checks pass, but the PIN in the plaintext does not match the PIN stored on the card, then the ICC PIN try counter is decremented and an error `0x63C`x is returned, where x is the new value of the PIN try counter. If all the checks succeed and the recovered PIN is correct, the card returns `9000`.

In our attack, each decryption attempt is highly likely to result in an error code being returned by the ICC. This is because, in the attack, the probability that the recovered ICC Unpredictable Number matches the ICC's generated value at step 7 is only 2^{-64}. In order for our attack to proceed at all, we have to make the assumption that the attacker can distinguish a failure at step 7 from a failure at step 8. We also have to assume that step 8 is either carried out before step 7, or is performed irrespective of whether step 7 is successful. These conditions might be met depending on the exact details of how the ICC's decryption procedure is implemented. We note that EMV's CPA specification [12], which specifies a "profile" for EMV cards, does provide more detail on how decryption processing should be performed in order to be compliant with that specification. In particular [12, Section 12.7, Requirements 12.30 and 12.31] and the update in [18] make it clear that step 8 is not carried out if step 7 fails, so our attack would not work for CPA-compliant cards.

The fact that each decryption attempt is highly likely to fail means that the attack can proceed without the risk of the card locking because of the PIN try counter reaching its limit. But cards may also keep a separate PIN decryption failure counter in addition to the PIN try counter – for example, this is an optional feature in the CPA specification [12]. However this counter and its use are not specified anywhere in the base EMV standards. Even if it is implemented for a particular card, its maximum value may be quite large to cater for bad terminal implementations. For example, in the CPA specification, it is a 2-byte counter, potentially allowing as many as 2^{16} failed decryption attempts. So an attacker may be able to make many decryption queries in an attack, and possibly without any limit.

Integration into the EMV CDA protocol: In an offline CDA transaction the transaction terminates with the card producing a summary of the transaction data (called a Transaction Certificate, TC) which is digitally signed by the card. The card's view of the cardholder verification method used during the transaction is normally present in the TC but in a data format that is proprietary to the issuer bank, and hence the terminal may not be able to check it. This weakness is exploited in the attacks of [26], through which an attacker may be able to carry out transactions using a lost or stolen EMV card without knowing its PIN. A natural fix to prevent these attacks for offline CDA transactions is to standardize the data format so that the terminal can verify that the cardholder verification method reported by the card in the TC matches its view of which method was used. Our attack below shows that, even with this fix in place, an attacker may still be able to complete offline CDA transactions without knowing the card's PIN. Before explaining the attack, we first illustrate how an EMV transaction would typically proceed. To simplify matters we focus mainly on the salient events that occur during an offline transaction where both the terminal and the card support CDA.

Offline CDA transaction processing: When a card is inserted into a terminal, the terminal first requests a list of supported applications. The terminal then selects an application from the list and starts a transaction. An EMV transaction progresses over three phases: card authentication, cardholder verification, and transaction

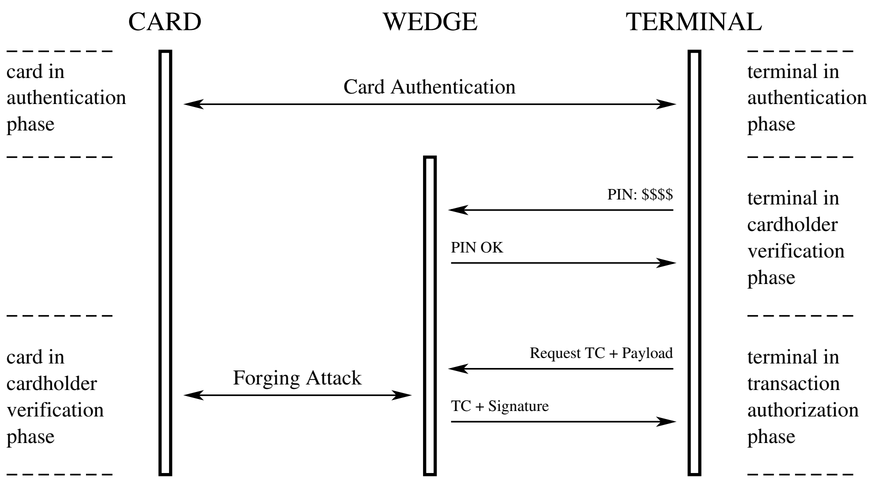
CARD
WEDGE
TERMINAL
card in authentication phase
Card Authentication
terminal in authentication phase
PIN: $$$$
PIN OK
terminal in cardholder verification phase
card in cardholder verification phase
Forging Attack
Request TC + Payload
TC + Signature
terminal in transaction authorization phase

Fig. 1. Executing an offline CDA transaction without the cardholder's PIN

authorization. Card authentication starts with a READ RECORD command issued by the terminal to retrieve the card details and other data. Included in the records is the card's RSA public key together with a certificate chain linking the card's private key to a card scheme root key known to the terminal. The terminal then requests the card to sign a provided nonce value. The terminal verifies the signature on the nonce through the card's public key which it in turn authenticates via the supplied certificate chain.

Next is cardholder verification where commonly the cardholder is required to enter his PIN through the terminal's keypad. When the card and terminal both support PIN encryption, the PIN will be encrypted at the terminal and transmitted to the card in encrypted form. PIN verification is initiated by the terminal requesting the PIN try counter from the card. This indicates the number of PIN entry attempts left before the card locks. If the PIN is to be encrypted the terminal also issues a GET CHALLENGE command to retrieve the 8-byte ICC Unpredictable number to be included in the encryption of the PIN block. The terminal then encrypts the PIN block under the card's public key and submits it to the card for verification through the VERIFY command. The card will recover the plaintext using its private key and carry out the decryption checks described above, returning either an error message or the 9000 code that indicates successful PIN verification.

Once the cardholder has been successfully verified, the terminal has to decide whether the transaction requires online authorization from the issuer bank or not. In the latter case, it requests the card to authorize the transaction. A GENERATE AC command is sent to the card, containing transaction details and a nonce generated by the terminal. If the card authorizes the transaction, it will respond with a Transaction Certificate (TC) cryptogram. Alternatively it can request the terminal to contact the issuer bank through an Authorisation Request Cryptogram (ARQC), or reject the transaction by responding with an Application Authentication Cryptogram (AAC) which aborts the transaction. The TC contains the transaction details authenticated via a MAC tag (included in the TC) that is computed using a symmetric key which the card and the issuer bank share. The terminal would normally keep a copy of the TC in case there is a dispute, and send a batch of the TCs to the acquiring bank at the end of day for clearing. Note that the MAC tag in the TC can only be verified by the issuer bank. Consequently offline transactions are susceptible to wedge attacks. A CDA card prevents such attacks by additionally providing the terminal with a signature computed over the TC, enabling the terminal to verify the authenticity of the TC offline.

Our offline CDA attack: The attack requires a device that intercepts and manipulates the communication between the card and the terminal. This could for instance be accomplished through a wedge device (a slim device that is inserted between the card and the terminal) or a fake card which relays communication to the real card as described in [26]. The attack is outlined in Figure 1, showing how the wedge interferes with the normal transaction flow. During the card authentication phase the wedge behaves passively and merely forwards messages between the terminal and the card. Once the terminal initiates the PIN verification phase, the wedge takes over the role of the card. The wedge can return any arbitrary value in response to the `GET CHALLENGE` command. When issued with the `VERIFY` command the wedge will indicate that the PIN verified correctly with a `9000` message, irrespective of the actual PIN value. Thus the attacker can enter any value on the terminal's PIN pad. Since the cardholder verification completed successfully the terminal will go on to request the card to authorize the transaction. At this point the wedge will extract the transaction details and the nonce from the authorization request (`GENERATE AC` command) and compose a TC in the same way that the card would. However, the wedge does not know the card's symmetric key in order to compute the MAC tag that should be present in this TC. Instead, the wedge just assigns a random value to the MAC tag. Since the transaction will be authorized offline, this defect will not be detected until later by the issuer bank.

The wedge now obtains a signature for this TC by mounting the forgery attack of Section 2 against the card. More specifically, the wedge will now impersonate the terminal to the card and initiate a series of PIN verification requests. The card will serve as the validity checking oracle, with the encrypted PIN in the payloads of `VERIFY` commands being replaced by the attack ciphertexts. Prior to each PIN verification request the wedge may need to issue a `GET CHALLENGE` command (see Requirement 12.29 of [12]). Once the wedge has forged a signature over the TC, the wedge forwards the TC together with its signature to the terminal to complete the original transaction.

Impact and practical considerations: In principle, the attack described above enables a wedge device to forge a signature on a TC, which the offline terminal will accept as being valid and thus authorize the transaction. The problem will only come to light later, once the issuer bank tries to verify the MAC in the TC, by which time the attacker equipped with the wedge device and a stolen card, will have made his escape with the purchased goods.

We stress that we have not implemented the above attack. There are several factors that may prevent it in practice. These include the fact that PIN encryption is not yet widely enabled, the fact that cards may use PIN decryption failure counters, the possibility of transaction time-outs being triggered because of the amount of time needed to produce the signature forgery, and the possibility that the `7F` oracle may not be available because of the way in which decryption is implemented (especially for CPA-compliant cards). Nevertheless, the attack illustrates the potential problems that may arise through reusing a keypair in different cryptographic operations.

4 Security Analysis of Combined Encryption and Signature for Elliptic Curve Algorithms

The prior sections detailed possible attacks when re-using the same key for encryption and signature in the existing EMV Co standards. But what can we say about upcoming versions of the standards? EMV Co has indicated that elliptic curve based algorithms are likely to be adopted in future versions of the EMV standards[3]. In particular PIN encryption will be performed by the public key algorithm ECIES [21], whilst digital signatures will be produced using either EC-DSA [20] or EC-Schnorr [22]. Before proceeding it is worth first recapping on what is known about the security of these three algorithms when used on their own, i.e. when used without sharing key-pairs.

ECIES is based on the DHIES encryption scheme [1]. In essence ECIES uses a one-sided static Diffie–Hellman key exchange to obtain a shared secret which is then combined with a one-time IND-CCA secure symmetric encryption scheme via the KEM-DEM paradigm. There are various "options" for use of ECIES in terms of how the agreed Diffie–Hellman key is used to obtain the shared secret. These variants are needed to deal with the well-documented benign malleability of ECIES when used in traditional mode. We use IND-gCCA to denote a scheme which is IND-CCA secure up to benign malleability [2].

[3] See `http://www.emvco.com/specifications.aspx?id=160` for draft specifications.

The known results for ECIES are that in traditional mode it is IND-gCCA secure in both the random oracle model [1] and in the generic group model [32]. Other variants, as defined in [21], can be shown to be IND-CCA secure. A full description of the various known results can be found in [10].

Security of EC-DSA is more problematic. The only known proof with respect to a relatively well-studied assumption is that it is secure, in the usual EUF-CMA sense, in the generic group model (GGM) with certain requirements on the hash function [5]. This proof makes crucial use of the conversion function f which maps elliptic curve points in $E(\mathbb{F}_p)$ to elements of $\mathbb{F}_q$ (note $q \neq p$). Brown [5] also gives a sketch proof under the assumptions that the hash function is a random oracle and that the group and conversion function f define a semi-logarithm problem which is intractable. We also note that EC-DSA is known to be insecure if used with a hash function for which collisions can be found.

Security of EC-Schnorr is much better understood. It is a classic result of Pointcheval and Stern [30] that Schnorr signatures are secure in the random oracle model, assuming the discrete logarithm problem is hard. In addition EC-Schnorr has been proved secure in the GGM, under the assumption that the hash function used meets further well-defined properties [27]. In particular, even if general collisions can be found in the hash function used in EC-Schnorr, the signature scheme is still secure. Thus EC-Schnorr is resistant to collision attacks on hash functions. Paillier and Vernaud [28] have argued that the above security results in idealized models are probably the best we can hope for, by providing evidence that it is unlikely that a security proof for Schnorr signatures in the standard model will be forthcoming.

In the rest of this section we study the joint security of ECIES and the signature schemes EC-DSA and EC-Schnorr, that is, the security of these schemes when the same key-pair is used for both signature and encryption. We show that all known security results carry over to the joint setting. Hence, if EMV Co allow the use of a single key-pair for their elliptic curve based algorithms (as they currently do for RSA-based schemes), there are no negative security implications: they would still obtain the strong security guarantees described above, just as if they had used two distinct key pairs.

4.1 Security Models for Joint Security

We first recall the security notions from [29] for combinations of a signature scheme and a public key encryption scheme that shares the same keypair (pk, sk). Such a combined signature and encryption scheme consists of a tuple of algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Enc}, \mathsf{Dec})$ where $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ form a signature scheme and $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ form a PKE scheme. The following security notions extend the standard notions of EUF-CMA and IND-CCA security by giving the adversary additional access to a signature (for IND-CCA) and decryption (for EUF-CMA) oracle that can be queried under the same challenge public key.

EUF-CMA security in the presence of a decryption oracle: Let $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Enc}, \mathsf{Dec})$ be a combined signature and encryption scheme. Existential unforgeability under an adaptive chosen message attack in the presence of additional oracles of the signature component is defined through the following game between a challenger and an adversary $\mathcal{A}$.

Setup: The challenger generates a keypair $(pk, sk) \leftarrow \mathsf{KGen}(1^k)$ and gives $\mathcal{A}$ the challenge public key pk.

Query phase: $\mathcal{A}$ requests signatures on messages m_i of its choice. The challenger responds to each signature query with a signature $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$. The adversary $\mathcal{A}$ can also request decryptions of ciphertexts c_i of its choice. The challenger responds to each decryption query with a message $m \leftarrow \mathsf{Dec}(sk, c_i)$ or a failure symbol $\perp$.

Forgery: $\mathcal{A}$ outputs a message signature pair (σ, m) such that m was not submitted to the signing oracle, and wins the game if $\mathsf{Verify}(pk, \sigma, m) = 1$.

The advantage of an adversary $\mathcal{A}$ is the probability it wins the above game. A forger $\mathcal{A}$ is said to (t, q_d, q_s, ϵ)-break the signature component of a combined signature and encryption scheme if $\mathcal{A}$ runs in time at most t, makes at most q_d decryption queries and q_s signature queries and has advantage at least ϵ. The signature component of a combined signature and encryption scheme is said to be (t, q_d, q_s, ϵ)-EUF-CMA secure in the presence of a decryption oracle if no forger (t, q_d, q_s, ϵ)-breaks it.

IND-CCA security in the presence of a signing oracle: Let $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Enc}, \mathsf{Dec})$ be a combined signature and encryption scheme. Indistinguishability under an adaptive chosen ciphertext attack in the presence of additional oracles of the encryption component is defined through the following game between a challenger and an adversary $\mathcal{A}$.

Setup: The challenger generates a keypair $(pk, sk) \leftarrow \mathsf{Keyen}(1^k)$ and gives $\mathcal{A}$ the challenge public key pk.

Phase 1: $\mathcal{A}$ requests decryptions of ciphertexts c_i of its choice. The challenger responds to each decryption query with a message $m \leftarrow \mathsf{PKE.Dec}(sk, c_i)$ or a failure symbol $\perp$. The adversary $\mathcal{A}$ can also request signatures on messages m_i of its choice. The challenger responds to each signature query with a signature $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$.

Challenge: $\mathcal{A}$ chooses two equal length messages m_0, m_1. The challenger chooses a random bit b, computes $c^* \leftarrow \mathsf{PKE.Enc}(pk, m_b)$, and passes c^* to the adversary.

Phase 2: As Phase 1 but with the restriction that $\mathcal{A}$ must not request the decryption of the challenge ciphertext c^*.

Guess: $\mathcal{A}$ outputs a guess b' for b.

The advantage of $\mathcal{A}$ is $\left|\Pr(b' = b) - \frac{1}{2}\right|$. An adversary $\mathcal{A}$ is said to (t, q_d, q_s, ϵ)-break the encryption component of a combined signature and encryption scheme if $\mathcal{A}$ runs in time at most t, makes at most q_d decryption queries and q_s signature queries and has advantage at least ϵ. The encryption component of a combined signature and encryption scheme is said to be (t, q_d, q_s, ϵ)-IND-CCA secure in the presence of a signing oracle if no adversary (t, q_d, q_s, ϵ)-breaks it.

Informally, we say that a combined scheme is *jointly secure* if it is both EUF-CMA secure in the presence of a decryption oracle and IND-CCA secure in the presence of a signing oracle.

4.2 Extending the KEM-DEM paradigm to combined schemes

To simplify the analysis of combined schemes where the encryption scheme is based on the KEM-DEM paradigm (such as ECIES) we adapt the security models from above to the KEM-DEM setting. We start by defining IND-CCA security for a KEM in the presence of a signing oracle that also uses the challenge public key. Further, we restate the KEM-DEM composition theorem with respect to such an additional signature oracle.

A key encapsulation mechanism $\mathsf{KEM} = (\mathsf{KEM.KGen}, \mathsf{KEM.Enc}, \mathsf{KEM.Dec})$ with associated keyspace $\mathcal{K}$ consist of a key generation algorithm $(pk, sk) \leftarrow \mathsf{KEM.KGen}(1^k)$; an encapsulation algorithm $(K, C) \leftarrow \mathsf{KEM.Enc}(pk)$ which returns a key $K \xleftarrow{\$} \mathcal{K}$ together with a ciphertext C; and a decryption algorithm $\{K, \perp\} \leftarrow \mathsf{KEM.Dec}(sk, C)$.

We now augment the notion of indistinguishability for KEM's [9] to our joint security setting by granting the adversary additional access to a signature oracle that uses the same secret key.

KEM-IND-CCA security in the presence of a signing oracle: Let $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{KEM.Enc}, \mathsf{KEM.Dec})$ be a combined signature and KEM scheme. Indistinguishability under an adaptive chosen ciphertext attack in the presence of additional oracles of the KEM component is defined through the following game between a challenger and an adversary $\mathcal{A}$.

Setup: The challenger generates a keypair $(pk, sk) \leftarrow \mathsf{KGen}(1^k)$ and gives $\mathcal{A}$ the challenge public key pk.

Phase 1: $\mathcal{A}$ requests decryptions of ciphertexts C_i of its choice. The challenger responds to each decryption query with decapsulated key $K \leftarrow \mathsf{KEM.Dec}(sk, C_i)$ or a failure symbol $\perp$. The adversary $\mathcal{A}$ can also request signatures on messages m_i of its choice. The challenger responds to each signature query with a signature $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$.

Challenge: When $\mathcal{A}$ queries the challenge oracle, the challenger chooses a random bit b, computes $(K^*, C^*) \leftarrow \mathsf{KEM.Enc}(pk)$, and passes (K^+, C^*) to the adversary, where $K^+ := K^*$ for $b = 0$ and $K^+ \xleftarrow{\$} \{0,1\}^k$ for $b = 1$.

```
$\mathsf{PKE.KGen}(1^k)$:
    $(pk, sk) \leftarrow \mathsf{KEM.KGen}(1^k)$
    return $(pk, sk)$

$\mathsf{PKE.Enc}(pk, M)$:
    $(K, C_1) \leftarrow \mathsf{KEM.Enc}(pk)$
    $C_2 \leftarrow \mathsf{DEM.Enc}(K, M)$
    return $(C_1, C_2)$

$\mathsf{PKE.Dec}(sk, C_1, C_2)$:
    $K \leftarrow \mathsf{KEM.Dec}(sk, C_1)$
    $M \leftarrow \mathsf{DEM.Dec}(K, C_2)$
    return $M$
```

Fig. 2. Building a PKE scheme from a KEM and a DEM.

Phase 2: As Phase 1 but with the restriction that $\mathcal{A}$ must not request the decryption of the challenge ciphertext C^*.
Guess: $\mathcal{A}$ outputs a guess b' for b.

The advantage of $\mathcal{A}$ is $\left|\Pr(b' = b) - \frac{1}{2}\right|$. An adversary $\mathcal{A}$ is said to (t, q_d, q_s, ϵ)-break the KEM-encapsulation component of a combined signature and KEM scheme if $\mathcal{A}$ runs in time at most t, makes at most q_d decryption queries and q_s signature queries and has advantage at least ϵ. The encryption component of a combined signature and KEM scheme is said to be (t, q_d, q_s, ϵ)-IND-CCA secure in the presence of a signing oracle if no adversary (t, q_d, q_s, ϵ)-breaks it.

A data encapsulation mechanism $\mathsf{DEM} = (\mathsf{DEM.Enc}, \mathsf{DEM.Dec})$ with keyspace $\mathcal{K}$ is a symmetric encryption scheme, that consist of an encryption algorithm $C \leftarrow \mathsf{DEM.Enc}(K, M)$; and a decryption algorithm $M \leftarrow \mathsf{DEM.Dec}(K, C)$. Security of a DEM is expressed as being secure against one-time CCA attacks.

IND-OTCCA security Let $(\mathsf{DEM.Enc}, \mathsf{DEM.Dec})$ be a DEM scheme. Indistinguishability under an one-time adaptive chosen ciphertext attack is defined through the following game between a challenger and an adversary $\mathcal{A}$.

Setup: The adversary $\mathcal{A}$ gets the security parameter 1^k as input.
Challenge: When $\mathcal{A}$ queries the challenge oracle on two equal length messages m_0, m_1, the challenger chooses a random bit b, samples a random key $K \overset{\$}{\leftarrow} \mathcal{K}$ and returns $C^* \leftarrow \mathsf{DEM.Enc}(K, m_b)$.
Decryption $\mathcal{A}$ can request decryptions of ciphertexts $C_i \neq C^*$ of its choice, with the restriction that it must not query the challenge ciphertext. The challenger responds to each decryption query with $M_i \leftarrow \mathsf{DEM.Dec}(K, C_i)$, using the same key K as in the challenge phase; or with a failure symbol $\perp$.
Guess: $\mathcal{A}$ outputs a guess b' for b.

The advantage of $\mathcal{A}$ is $\left|\Pr(b' = b) - \frac{1}{2}\right|$. A DEM scheme is said to be (t, q, ϵ)-IND-OTCCA if no adversary (t, q, ϵ)-breaks it, where $\mathcal{A}$ runs in time at most t, makes at most q decryption queries and has advantage at least ϵ.

Given a KEM and a DEM we can obtain a full-fledged PKE as shown in Figure 2. It was shown in [9] that by composing a KEM-IND-CCA secure key encapsulation mechanism and a IND-OTCCA secure encryption scheme one obtains a IND-CCA secure public key encryption scheme. We now show a similar result with respect to an additional signing oracle that uses the same key generation, and thus the same key-pair as the KEM. That is, we show that if the KEM scheme is KEM-IND-CCA secure in the presence of a signing oracle, and the DEM is IND-OTCCA secure, then the composed scheme is also IND-CCA secure in the presence of a signing oracle.

Theorem 1. *Let KEM be an IND-CCA secure KEM scheme in the presence of a signing oracle and let DEM be an IND-OTCCA secure encryption scheme. Then the composed KEM-DEM scheme is IND-CCA secure in the presence of a signing oracle.*

Proof. (sketch) The proof follows the argumentation of [9] closely: In particular we proceed via a sequence of games. Game 0 is the standard game as in the definition above. Then in Game 1 we modify the challenge oracle, such that $\mathsf{PKE.Enc}$ now uses a random key $K^{\$}$ to obtain C_2^* rather than the real key that is encapsulated in C_1^*. Thus, the challenge ciphertext of M_b was produced using a random key, which is independent from C_1^* or any other value in Game 1, which allows to turn any successful adversary in Game 1 into an adversary which breaks the OTCCA security of the data encapsulation scheme DEM. If the change from Game 0 to Game 1 results in a noticeable difference of $\mathcal{A}$'s success probability, this allows to construct a successful adversary against the KEM-IND-CCA security of the combined signature and KEM scheme.

Note that the statement above is easily adapted to the case where the KEM part is only IND-gCCA secure, with the obvious difference that the obtained PKE achieves only IND-gCCA security too.

4.3 ECIES, EC-Schnorr and EC-DSA in a Nutshell

In this section we briefly describe ECIES, EC-DSA and EC-Schnorr schemes according to their respective ISO specifications.

ECIES Encryption Scheme

ECIES is a public-key encryption scheme based on elliptic curves which follows the KEM-DEM paradigm, i.e., it consists of a key encapsulation scheme $\mathsf{ECIES\text{-}KEM}$, and a data encapsulation scheme DEM which uses one-time pad encryption in combination with HMAC [13,21]. Let (E, G, q) be system parameters specifying a secure elliptic curve E together with a point G that generates a secure cyclic subgroup of E, with prime order q, let $\mathsf{KDF} : \{0,1\}^m \rightarrow \{0,1\}^*$ be a key-derivation function, $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^{\ell_H}$ be a hash function and $\mathsf{HMAC}_{\mathcal{H}}$ be the HMAC-transform of $\mathcal{H}$.
Key Generation. $\mathsf{ECIES\text{-}KEM.KGen}$ gets as input the system parameters (E, G, q), chooses a random integer $x \xleftarrow{\$} \{1, \ldots, q-1\}$, sets $Y := xG$ and outputs the keys $pk = (E, G, q, Y)$ and $sk = x$.
Key Encapsulation. $\mathsf{ECIES\text{-}KEM.Enc}$ on input the public key pk and a message length ℓ_M chooses a random value $r \xleftarrow{\$} \{1, \ldots, q-1\}$, sets $C := rG$ and $K := \mathsf{KDF}([rY]_x)$ where $[rY]_x$ is the x coordinate of rY. The key-derivation function KDF is specified as

$$\mathsf{KDF}(m) = \mathcal{H}(m \| \langle 1 \rangle_{32}) \;\; \| \;\; \mathcal{H}(m \| \langle 2 \rangle_{32}) \;\; \| \;\; \mathcal{H}(m \| \langle 3 \rangle_{32}) \;\; \| \;\; \ldots \mathcal{H}(m \| \langle n \rangle_{32})$$

for $n = (\ell_M + \ell_H/2)/\ell_H$ and where $\langle i \rangle_{32}$ denotes the integer i in binary encoding, represented with 32 bits. It outputs the encapsulated key as (C, K).
Key Decapsulation. $\mathsf{ECIES\text{-}KEM.Dec}$ on input the public key $sk = x$ and an encapsulated key C first computes $Q := xC$ and checks whether $Q \neq 0$. If so, it outputs $K := \mathsf{KDF}(Q_x)$ where Q_x is the x coordinate of Q and "fail" otherwise.
Data Encapsulation. $\mathsf{DEM.Enc}$ on input a message M and a key K first parses the key as $K = K_1 \| K_2$. It then computes $C := M \oplus K_1$ and a tag $t := \mathsf{HMAC}_{\mathcal{H}}(K_2, C)$ and outputs (C, t)
Data Decapsulation. $\mathsf{DEM.Dec}$ on input a ciphertext (C, t) and a key K parses the key as $K = K_1 \| K_2$ again. It then computes $M := C \oplus K_1$ and verifies whether $t = \mathsf{HMAC}_{\mathcal{H}}(K_2, C)$. If the verification fails, it returns $\perp$ and M otherwise.
The suggested KEM variant achieves IND-gCCA security only, as it hashes only the x-coordinate of rY instead of the full point which allows for a simple — but in practice harmless – attack against the full-fledged CCA security. Note also that the DEM above is easily malleable if variable-length messages are allowed [31]. However, as the public-key encryption will be applied only to fixed-length messages (containing variable length PINs), the security of the proposed DEM is sufficient.

EC-Schnorr Signature Scheme

Let (E, G, q) be again system parameters specifying a secure elliptic curve E together with a generator point G that generates a secure cyclic subgroup $\mathbb{G}$ with prime order q.

Key Generation. EC-Schnorr.KGen gets as input the system parameters (E, G, q), chooses a random integer $x \overset{\$}{\leftarrow} \{1, \ldots, q-1\}$, sets $Y = xG$ and outputs the keys $pk = (E, G, p, Y)$ and $sk = x$.

Signing. EC-Schnorr.Sign on input the secret key $sk = x$ and a message M chooses a random value $r \overset{\$}{\leftarrow} \{1, \ldots, q-1\}$, sets $R = rG$ and computes $h := \mathcal{H}(f(R_x) \| f(R_y) \| M)$, where R_x denotes the x-coordinate of R and $f(\cdot)$ is a conversion function that converts a field element into a bit-string. It further computes $s := r + hx \mod q$ and outputs the pair (h, s) unless $s = 0$ or $h = 0$. In this latter case the whole procedure is repeated.

Verification. EC-Schnorr.Verify on input the public key pk, message M and a signature (h, s) first verifies whether $h \neq 0$ and lies in the domain of the hash function and if $s \in \{1, \ldots, q-1\}$. If one of the conditions is not fulfilled it outputs "invalid", otherwise it continues the verification and computes R' as $R' := sG - hY$ and $h' := \mathcal{H}(f(R'_x) \| f(R'_y) \| M)$. If $h' = h$ it outputs "valid" and "invalid" otherwise.

EC-DSA Signature Scheme

Let (E, G, q) be again system parameters specifying a secure elliptic curve E together with a generator point G that generates a secure cyclic subgroup with prime order q.

Key Generation. EC-DSA.KGen gets as input the system parameters (E, G, q), chooses a random integer $x \overset{\$}{\leftarrow} \{1, \ldots, q-1\}$, sets $Y = xG$ and outputs the keys $pk = (E, G, p, Y)$ and $sk = x$.

Signing. EC-DSA.Sign on input the secret key $sk = x$ and a message M chooses a random value $k \overset{\$}{\leftarrow} \{1, \ldots, q-1\}$, sets $R := kG$. If $R_x = 0$, where R_x denotes the x-coordinate of R, this step is repeated. Otherwise r is set to be the value of R_x mapped (via it's bit representation) to an element of $\mathbb{F}_q$. The signer then computes $h := \mathcal{H}(M)$ and $s := (h + r \cdot x)/k \mod q$ and outputs the pair (r, s), unless $s = 0$ or $r = 0$. In this latter case the whole procedure is repeated. For future reference the function which sends R to r used in signing is referred to as the "conversion function", we write $r := f(R)$.

Verification. EC-DSA.Verify on input the public key pk, message M and a signature (r, s) first verifies whether r and s lie in the range $\{1, \ldots, q-1\}$. The value $h := \mathcal{H}(M)$ is computed and the verifier computes $a := h/s \mod q$ and $b := r/s \mod q$. Then the value $R' := aG + bY$ is computed, and the signature is accepted if and only if $r = f(R')$.

4.4 On the Joint Security of ECIES and EC-Schnorr

We first sketch why ECIES and EC-Schnorr are jointly secure in the ROM. By Theorem 1 we need only show that ECIES-KEM and EC-Schnorr are jointly secure. Thus in the signature game for EC-Schnorr we need to simulate the ECIES decapsulation queries, and in the security game for ECIES-KEM we need to simulate the signing queries for EC-Schnorr. We do not present the proofs in full detail, but simply explain how these extra simulations can be incorporated into the existing proofs.

Security of the KEM Component in the ROM. We start with the simpler case of showing that one can answer EC-Schnorr signature queries within the ECIES-KEM security proof without sacrificing security. Roughly, the ECIES-KEM proof from [1, 9] reduces the IND-(g)CCA security to the gap-Diffie-Hellman problem by embedding a DH challenge into the public key and the challenge ciphertext. Recall that the symmetric key in ECIES-KEM is derived from applying the key derivation function KDF on a "Diffie-Hellman key". It is shown that if the KDF is assumed to be a random oracle, a successful adversary against the ECIES-KEM needs to query the DH-key — and thus the solution to the DH challenge — to the random oracle. In our joint setting we have to reduce this to assuming that the hash function $\mathcal{H}$, that is used to construct the KDF, is a random oracle instead. This stems from the fact that the signature component makes use of a hash function as well and we only want to assume a single random oracle. However, it is easy to see that the KDF as described above inherits the random oracle property from the underlying hash function. We further see that every call to $\mathcal{H}$ is then of the form $P_x \| \langle i \rangle_{32}$, where i is an integer and P_x is a point on the elliptic curve, which allows to adapt the argumentation from the original proof. Note that the DHIES proof in [1] needs to be slightly modified to the case of ECIES due to the benign malleability for the standardized variant of ECIES-KEM mentioned above, but this is a trivial exercise.

To simulate EC-Schnorr queries within this proof we run the "standard" signature simulation from the forking-lemma proof of Schnorr signatures [30]; i.e. the simulator generates h and s at random in $\{1, \ldots, q-1\}$, then defines $R = sG - hY$ and then "patches" the random oracle so that $h = \mathcal{H}(R_x \| R_y \| M)$, where M is the message being signed. One immediately sees that the input to the oracle in this simulation will never interfere with the input to the oracle for the decapsulation queries, and vice-versa. That is, an adversary can not exploit the signature oracle to obtain decryptions of encapsulated keys. Thus the simulation of the signing oracle is perfect, in the random oracle model.

Hence, security of ECIES-KEM in the presence of an EC-Schnorr signing oracle is guaranteed, as long as the conditions for the proof of ECIES-KEM are satisfied. This is that the gap-Diffie–Hellman (gap-DH) problem is hard; i.e. an adversary cannot solve the Diffie–Hellman problem even when given access to a decision Diffie–Hellman (DDH) oracle.

Security of the Signature Component in the ROM. Security of EC-Schnorr in the presence of an ECIES decapsulation oracle follows in much the same way. We simply need to modify the standard forking-lemma based proof of Schnorr signatures so that the adversary in addition has a decapsulation oracle for ECIES-KEM. In the proof the decapsulation queries are simply answered by using the decapsulation simulator from the proof of ECIES-KEM in the random oracle model. Again this latter simulation usually treats KDF as the random oracle, but this is easily replaced by assuming $\mathcal{H}$ is a random oracle instead. Again we also see that the two uses of the random oracle are compatible with each other, due to the sizes of the input values. Thus, we can run the simulations of the hash function for the EC-Schnorr proof and for the ECIES-KEM part in parallel (as the simulator can easily detect if the adversary uses the RO in the context of the signature or the ECIES-KEM component). However, to ensure the hash function queries for the ECIES component are answered correctly the simulator will need access to an oracle which solves DDH. Thus whilst EC-Schnorr is secure assuming the DLP problem is hard, the scheme is only jointly secure assuming DLP is hard even when given access to a DDH oracle. We call this the gap-DLP problem (by analogy with the more standard gap-DH problem mentioned above).

Putting these two informal arguments together, we have:

Theorem 2. *In the random oracle model ECIES-KEM and EC-Schnorr are jointly secure if the gap-DLP problem and gap-DH problem are both hard.*

By adapting the method in the following section one can also show that ECIES-KEM and EC-Schnorr are jointly secure in the GGM if the hash function is random-prefix (second-)preimage resistant and the conversion function f is partially invertible and uniform. This is done by combining the proof in the GGM below for ECIES-KEM with the proof of EC-Schnorr in the GGM found in [27]. We refer to [27] for a definition of what it means for a hash function to be random-prefix (second-)preimage resistant.

4.5 On the Joint Security of ECIES and EC-DSA

We now sketch why ECIES-KEM in combination with EC-DSA are jointly secure in the generic group model (GGM). The idea of the generic group model is that an adversary can not exploit any concrete feature of the group, i.e., the group is considered to be ideal. This is modeled by giving the adversary only oracle — and thus indirect — access to the group elements and group operations. That is, for any input $i \in \mathbb{Z}_q$ the oracle will return the representation $\tau(i) \in \mathbb{G}$, and for any query $(\tau(i), \tau(j), a, b)$ it responds with the element $\tau(ai + bj)$. Note that for the latter query one does not necessarily have to know i or j. To ensure that the oracle is the only way to perform operations on the representations of the group elements, the encoding function τ is chosen randomly from the set of all possible mappings from $\mathbb{Z}_q \to \mathbb{G}$.

Similar to the ROM proof discussion above, we show that one is able to simulate the additional decryption or signing oracle while retaining the original proofs for EC-DSA or ECIES in the GGM. Again, we focus on showing how to handle the extra simulations in the existing proofs.

Security of the KEM Component in the GGM. For full ECIES (i.e. KEM+DEM), security in the GGM was shown under the DDH assumption (which trivially holds in the GGM) and the security of the DEM, but interestingly without any assumptions on the key derivation function. When switching to ECIES-KEM only, it however requires some uniformity property of the KDF as briefly mentioned by Shoup [31]. This property roughly says that the output of a KDF/hash function on a random (and secret) input can not be distinguished from a truly random value.

The proof for ECIES-KEM starts with a tiny game hop, where in the modified KEM-IND-CCA game the challenge ciphertext-key-pair for $b = 1$ is generated as $(rG, \mathsf{KDF}(zG))$ for a random $z \in \{1, \ldots, q-1\}$ instead of (rG, K^+) with K^+ being a random key. Due to the uniformity property of the KDF, this can not result in a noticeable change of $\mathcal{A}$'s success probability. For any successful adversary in the new game that breaks the KEM-IND-CCA property, we can now easily obtain an equally successful adversary breaking the DDH assumption, which can only happen with exponentially small probability in the GGM. That is on input $(\tau(x), \tau(y), \tau(z))$ an adversary has to output 0 if it believes that $z = xy$ and 1 otherwise. To this end, we embedded the DDH challenge in the public key and challenge ciphertext as $pk = \tau(x)$ and $C^* = (\tau(y), \mathsf{KDF}(\tau(z)))$. Thus, if z is a valid DH value, this corresponds to $b = 0$ where the key has the correct form $\mathsf{KDF}(\tau(xy))$ whereas it will correspond to $b = 1$ where a derivation on a random point is given otherwise.

To answer queries to the decryption, signing and group oracle in a consistent way, the simulator maintains a list $\mathcal{L}$ of tuples $(\tau(i), v, w)$ which denotes that the adversary had learned (either directly via a GG call or indirectly via a decryption/signing query) the representation $\tau(i)$ of $i = v + w \cdot x$. For any query $(\tau(i), \tau(j), a, b)$ to the group operation oracle, where $\tau(i), \tau(j)$ must be encodings from previous queries, the simulator first checks if $\mathcal{L}$ contains an entry $(\tau(k), av_i + av_j, bw_i + bw_j)$. If so, it returns $\tau(k)$, otherwise it chooses a random representation $\tau(k) \in \mathbb{G}$ which is different from all other elements stored in $\mathcal{L}$, adds $(\tau(k), av_i + av_j, bw_i + bw_j)$ to its list and returns $\tau(k)$.

The list is further initialized with the tuples $(\tau(1), 1, 0)$ for the generator of the group and $(\tau(x), 0, 1)$ for the public key, where $\tau(1)$ is randomly chosen and $\tau(x)$ is the first value of the DDH challenge.

When responding to a decryption query $\tau(r)$, the simulator first obtains $(\tau(r), v, w)$, from its record and checks if $\mathcal{L}$ already contains an entry $(\tau(k), 0, v + w\bar{x})$. If not it chooses a random representation $\tau(k)$ and adds the tuple $(\tau(k), 0, v + w\bar{x}))$ to $\mathcal{L}$. Note that we can not evaluate $v + wx$ since we do not know x, thus we keep it is as a polynomial with $\bar{x}$ denoting a variable. The decryption oracle finally returns $K \leftarrow \mathsf{KDF}(\tau(k))$.

To simulate signature queries the oracle on input of some message M, chooses a random element $\tau(k)$, computes r, h according to the EC-DSA signing algorithm, chooses $s \in \{1, \ldots, q-1\}$ at random and returns (r, s) to the adversary. It further adds the tuple $(\tau(k), h/s, r/s)$ to $\mathcal{L}$. Thus, when an adversary wants to verify a signature (r, s, M) it must obtain the value $\tau(h/s + rx/s)$ from its group oracle which will then respond with $\tau(k)$ again.

Further, as long as no two entries $(\tau(i), a_i, b_i)$, $(\tau(j), a_j, b_j)$ with $a_i + b_i\bar{x} = a_j + b_j\bar{x}$ but $\tau(i) \neq \tau(j)$ exist, the simulation of the decryption and signature oracle are perfect. The probability of this event can be upper bounded by $|\mathcal{L}|^2/q$.

Security of the Signature Component in the GGM. Brown proved in [4] that EC-DSA is secure in the generic group model if the conversion function f is almost invertible and the hash function is uniform, collision resistant and zero-finder-resistant. The proof mainly shows that each successful forgery (M^*, s^*, r^*) requires an entry $(\tau(k^*), \mathcal{H}(M^*)/s^*, r^*/s^*)$ where $r^* = f(\tau(k^*))$. Depending on the event that triggered this entry, reductions to the underlying hash function assumptions are given. To embed a message for the collision-finder the proof handles queries to the group oracle differently from the proof for IND-CCA security above. Namely, for a query $(\tau(i), \tau(j), a, b)$ where $(\tau(k), a_i + b_i, a_j + b_j)$ is not defined yet, it chooses a random message M and computes $\tau(k) \leftarrow f^{-1}(\mathcal{H}(M) \cdot (a_i + a_j)^{-1} \cdot (b_i + b_j))$. For this step the invertibility of f and the uniformity of h are required, as $\tau(k)$ should not leak information about M.

Queries M to the signing oracle are answered exactly as defined by the EC-DSA algorithm. Here the secret key x is chosen at random, but known to the simulator since we do not play against any computational assumption in the generic group. The simulator further adds for each query an entry $(\tau(k), \mathcal{H}(M)/s, r/s)$ to $\mathcal{L}$. To handle the additional decryption queries, the simulator reacts as above, i.e., on input a ciphertext

$\tau(r)$, it obtains $(\tau(r), v, w)$, from its record and checks if $\mathcal{L}$ already contains an entry $(\tau(k), 0, v + wx)$ (recall that this time x is known). If not it chooses a random representation $\tau(k)$ and adds the tuple to $\mathcal{L}$. It finally returns $K \leftarrow \mathsf{KDF}(\tau(k))$. Due to the knowledge of x both simulations are perfect.

Brown showed that if the entry $(\tau(k^*), \mathcal{H}(M^*)/s^*, r^*/s^*)$ corresponding to the valid forgery was created by the group or signing oracle, this requires that either $\mathcal{H}(M^*) = 0$ or $\mathcal{H}(M^*) = \mathcal{H}(M)$ for one of the messages that was "embedded" in the group oracle responses. In our joint setting we have to consider the additional event that the entry was created by the decryption oracle. As all tuples created by the decryption oracle have a zero as second element that requires $\mathcal{H}(M^*) = 0$ which also contradicts the zero-finder-resistance of $\mathcal{H}$.

Putting both of the above arguments together we obtain:

Theorem 3. *In the generic group model ECIES-KEM and EC-DSA are jointly secure if the DDH problem is hard, the hash function $\mathcal{H}$ is uniform, collision-resistant and zero-finder-resistant and the conversion function f is almost invertible.*

5 Conclusions

Our results on RSA in EMV provide an illustration, should one still be needed, that the deployment of cryptographic algorithms having *ad hoc* designs not supported by formal security analysis can lead to potential and actual security weaknesses. This is especially true when the algorithms are used as components in complex protocols where the possible interactions between algorithms are many and therefore hard to assess.

While the key separation principle dictates using different keys for different cryptographic functions, there are performance benefits that may accrue from reusing keys in constrained environments, and the EMV standards allow key reuse for this reason. We have provided positive security results for the likely candidates for ECC-based algorithms in EMV when keys are re-used. Our results rule out large classes of attack, including attacks like those we exhibited for the existing RSA algorithms. However, we wish to close with a strong *caveat*: as a research community, we are nowhere close to having a full analysis of the EMV protocol suite. Its complexity and support for vendor variation means that it is likely to be beyond the reach of either provable security or formal methods techniques for some time to come. For example, the positive results for ECC-based algorithms in this paper do not rule out the type of protocol-level attack presented in [26]. However, our cryptographic analysis could help to form the foundation of such a larger-scale study.

6 Acknowledgements

All authors would like to acknowledge the partial support by the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II. The first author was also supported by Vodafone Group Services Limited, a Thomas Holloway Research Studentship, and the Strategic Educational Pathways Scholarship Scheme (Malta), part-financed by the European Union – European Social Fund. The third author was also supported by EPSRC Leadership Fellowship, EP/H005455/1. The fourth author was also supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, and in part by a Royal Society Wolfson Merit Award. The fifth author was also supported in part by the French ANR-07-TCOM-013-04 PACE Project.

We thank the EMVCo Security Working Group for reviewing an earlier version of this paper.

References

1. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, 2001.
2. Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 83–107, 2002.

3. Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS 1. In *CRYPTO '98*, volume 1462 of *LNCS*, pages 549–570. Springer, 1998.
4. Daniel Brown. Generic groups, collision resistance, and ECDSA. *Des. Codes Cryptography*, 35:119–152, 2005.
5. Daniel Brown. On the provable security of ECDSA. In G. Seroussi I.F. Blake and N.P. Smart, editors, *Advances in Elliptic Curve Cryptography*, pages 21–40. Cambridge University Press, 2005.
6. Jean-Sébastien Coron, Marc Joye, David Naccache, and Pascal Paillier. Universal padding schemes for RSA. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 226–241. Springer, 2002.
7. Jean-Sebastien Coron, David Naccache, and Mehdi Tibouchi. Fault attacks against EMV signatures. In *CT-RSA 2010*, volume 5985 of *LNCS*, 2010.
8. Jean-Sébastien Coron, David Naccache, Mehdi Tibouchi, and Ralf-Philipp Weinmann. Practical cryptanalysis of ISO/IEC 9796-2 and EMV signatures. In *CRYPTO 2009*, volume 5677 of *LNCS*, pages 428–444. Springer, 2009.
9. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33:167–226, 2003.
10. Alexander W. Dent. Proofs of security for ECIES. In G. Seroussi I.F. Blake and N.P. Smart, editors, *Advances in Elliptic Curve Cryptography*, pages 41–66. Cambridge University Press, 2005.
11. Yvo Desmedt and Andrew M. Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In *CRYPTO '85*, volume 218 of *LNCS*, pages 516–522. Springer, 1985.
12. EMV Co. EMV Common Payment Application Specification – Version 1.0, December 2005.
13. EMV Co. EMV Book 2 – Security and Key Management – Version 4.1z ECC – With support for Elliptic Curve Cryptography, May 2007.
14. EMV Co. EMV Book 1 – Application Independent ICC to Terminal Interface Requirements – Version 4.2, June 2008.
15. EMV Co. EMV Book 2 – Security and Key Management – Version 4.2, June 2008.
16. EMV Co. EMV Book 3 – Application Specification – Version 4.2, June 2008.
17. EMV Co. EMV Book 4 – Cardholder, Attendant, and Acquirer Interface Requirements – Version 4.2, June 2008.
18. EMV Co. EMV Specification Bulletin No. 84, December 2010.
19. Stuart Haber and Benny Pinkas. Securely combining public-key cryptosystems. In *ACM Conference on Computer and Communications Security*, pages 215–224, 2001.
20. ISO/IEC. ISO/IEC 14888-3:2006, Information technology – Security techniques – Digital signatures with appendix – Part 3: Discrete logarithm based mechanisms, 2006.
21. ISO/IEC. ISO/IEC 18033-2, Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers, 2006.
22. ISO/IEC. Final Draft of ISO/IEC 14888-3:2006, Information technology – Security techniques – Digital signatures with appendix - Part 3: Discrete logarithm based mechanisms – Amendment 1: Elliptic Curve Russian Digital Signature Algorithm, Schnorr Digital Signature Algorithm, Elliptic Curve Schnorr Digital Signature Algorithm, and Elliptic Curve Full Schnorr Digital Signature Algorithm, 2010.
23. David Naccache Jean-Sebastien Coron and Julien P. Stern. On the security of RSA padding. In *CRYPTO '99*, volume 1666 of *LNCS*, pages 1–18. Springer, 1999.
24. Vlastimil Klíma and Tomás Rosa. Further results and considerations on side channel attacks on RSA. In *CHES 2002*, volume 2523 of *LNCS*, pages 244–259. Springer, 2002.
25. Yuichi Komano and Kazuo Ohta. Efficient universal padding techniques for multiplicative trapdoor one-way permutation. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 366–382. Springer, 2003.
26. Steven J. Murdoch, Saar Drimer, Ross Anderson, and Mike Bond. Chip and PIN is broken. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pages 433–446, Oakland, CA, USA, May 2010.
27. Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for Schnorr signatures. *J. Mathematical Cryptology*, 3:69–87, 2009.
28. Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 1–20. Springer, 2005.
29. Kenneth G. Paterson, Jacob C.N. Schuldt, Martijn Stam, and Susan Thomson. On the joint security of encryption and signature, revisited. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 161–??? Springer, 2011.
30. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.
31. Victor Shoup. A proposal for an ISO standard for public key encryption (version 2.1). `http://www.shoup.net/papers/iso-2\_1.pdf`, 2001.
32. Nigel P. Smart. The exact security of ECIES in the generic group model. In *Coding and Cryptography*, volume 2260 of *LNCS*, pages 73–84, 2001.
33. Nigel P. Smart. Errors matter: Breaking RSA-based PIN encryption with thirty ciphertext validity queries. In *CT-RSA 2010*, volume 5985 of *LNCS*, pages 15–25. Springer, 2010.

A Graphs of Experimental Data

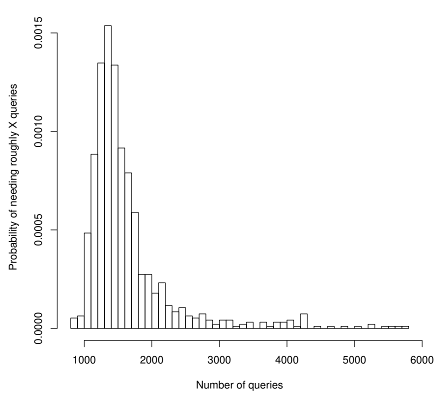

Fig. 3. Histogram of number of queries needed for 512 bit keys

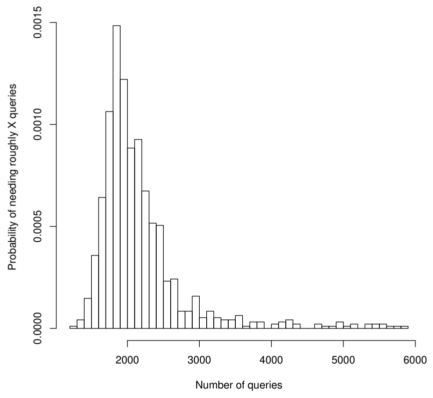

Fig. 4. Histogram of number of queries needed for 768 bit keys

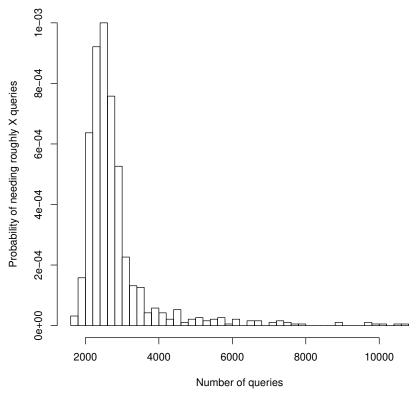

Fig. 5. Histogram of number of queries needed for 1024 bit keys

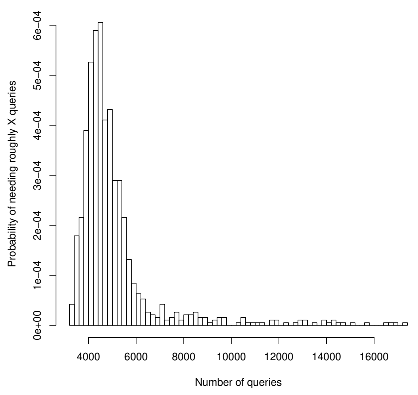

Fig. 6. Histogram of number of queries needed for 1984 bit keys