Relay Cost Bounding for Contactless EMV Payments

Tom Chothia[1], Flavio D. Garcia[1], Joeri de Ruiter[2], Jordi van den Breekel[3], and Matthew Thompson[1]

[1] School of Computer Science, University of Birmingham, UK
[2] Institute for Computing and Information Sciences, Radboud University Nijmegen
[3] Department of Mathematics and Computer Science, Technical University Eindhoven

Abstract. This paper looks at relay attacks against contactless payment cards, which could be used to wirelessly pickpocket money from victims. We discuss the two leading contactless EMV payment protocols (Visa's payWave and MasterCard's PayPass). Stopping a relay attack against cards using these protocols is hard: either the overhead of the communication is low compared to the (cryptographic) computation by the card or the messages can be cached before they are requested by the terminal. We propose a solution that fits within the EMV Contactless specification to make a payment protocol that is resistant to relay attacks from commercial off-the-shelf devices, such as mobile phones. This solution does not require significant changes to the cards and can easily be added to existing terminals. To prove that our protocol really does stop relay attacks, we develop a new method of automatically checking defences against relay attacks using the applied pi-calculus and the tool ProVerif.

1 Introduction

EMV is the most widely used standard for payments using smart cards [13]. The EMV Contactless specification has been introduced to support contactless smart cards [14]. For every payment provider a different variation of the specification exists. MasterCard and Visa market their solutions as PayPass and payWave (primarily the qVSDC protocol) respectively.

A typical attack against smart cards are so-called relay attacks, as demonstrated for EMV in [10]. Here an attacker uses a reader and card emulator to relay communication between a victim's card and a genuine terminal. This way it would, for example, be possible to pay for an expensive item in a different shop while the victim thinks the payment is only for an inexpensive item. With contact based smart cards the opportunity to use this attack is limited, as the victim knows when the card is being used: namely when it is inserted into a reader. For a contactless card this protection is lost, as the transaction does not require the users PIN number, it is enough to hold a reader close to a card to be able to communicate with it, even if it is inside a wallet or handbag. Therefore the attacker will have more opportunities to perform the attack and it will be less clear for the user that anything is going on. That such attacks are possible using cheap hardware, namely mobile phones, has been demonstrated in, for example, [17,11,12,20].

A typical solution to counter relay attacks are distance bounding protocols. Classic distance bounding protocols have been widely studied in the literature and elegant

solutions have been proposed [18,10,8,6]. These protocols assume that card and reader share a secret and then measure the time it takes to exchange a number of bits. Based on accurate time measurements at the level of nano seconds, knowledge of the clock speeds on both sides and the speed of light, an estimate can be made of the distance to the other party with an accuracy of a few meters. This assumes an attacker that is capable of relaying messages at close to the speed of light. While this is possible with specialised hardware we considered this attacker model to be an overkill for contactless EMV [15]. As contactless transactions can only be used for small amounts without a PIN, and the use of specialised equipment may raise suspicion (and so the chance of getting caught) such an attack offers a poor risk/reward ratio. A much better risk/reward ratio will come from using inconspicuous hardware that attackers may already own, such as NFC enabled smartphones. Another practical obstacle for classic distance bounding is that currently there is no shared secret between reader and card and incorporating such (complex) protocol would require a complete redesign of the existing infrastructure.

In this paper, we propose a protocol for contactless EMV payments, which will stop relay attacks using mobile phones, or off-the-shelf USB NFC readers. The protocol we propose is simpler than existing distance bounding protocols and is based on restricting the response time of a single message to the level of milliseconds. The solution is software based and does not require additional hardware to be added to existing terminals. It will not detect messages relayed at close to the speed of light, but we show that it will stop relay attacks that use inexpensive, easily available hardware.

We observe that in the current contactless protocols there are two types of message that need to be relayed from the reader to the card. The first type of message is used in payWave to retrieve a nonce used in the previous cryptographic operation. This nonce can already be retrieved while the terminal is still requesting other commands. The second type is one that requires the card to carry out some complex cryptographic computation. This is used in both payWave and PayPass. We have found that the variation in the time it takes different cards to carry out the cryptographic calculation is larger than the time needed for smartphones to relay the message. Therefore, imposing a time bound on the reply to either of these messages cannot be used to stop relay attacks. On the other hand, the time it takes cards to reply to messages that do not require cryptographic operations is low and uniform across all the cards we looked at. However, in the current protocols, these messages can all be cached by an attacker, and therefore cannot be time bound either.

The protocol we propose fits within the current EMV Contactless specification. A message exchange is introduced that must be relayed between the reader and the card, but does not require any cryptographic computation from the card and cannot be cached. Without the cryptographic operations the timing of this message will be more uniform for different cards, so we may use this message to add a timing bound and stop relay attacks.

To formally verify the correctness of the protocol we propose a new technique for modelling relay attacks using the applied pi-calculus [1] and using the tool ProVerif [3]. It would be possible to build a formal model which includes the time of each action and detect when a message takes too long, but such models would be complex and tool support is limited. Instead we assume that the designer of a system has identified

a message and a reply that the attacker will not be able to relay fast enough. Then, given this assumption, the formal verification checks if there are any messages that an attacker could pre-play, replay, or invent to break the protocol.

The contribution of this paper is:

- Showing that relay attacks are possible against MasterCard's PayPass protocol, (as well as Visa's payWave) using mobile phones that cache static messages.
- Proposing a contactless EMV protocol, which is a combination of the payWave and PayPass protocols, and which defends against relay attacks.
- Showing how these defences against relay attacks can be formally verified in the applied pi-calculus and automatically checked with the tool ProVerif.

Structure of the paper: we review the EMV and EMV Contactless protocol specifications in Section 2 and we look at the practicalities of a relay attacks in Section 3. We present our new protocol in Section 4. We formally verify this in Section 5 and discuss its implementation in Section 6. We conclude in Section 7. Our website[4] provides additional information, such as our timing measurements from cards, complete protocol traces, ProVerif models and a video of our relay in action.

2 EMV and the payWave/PayPass Protocols

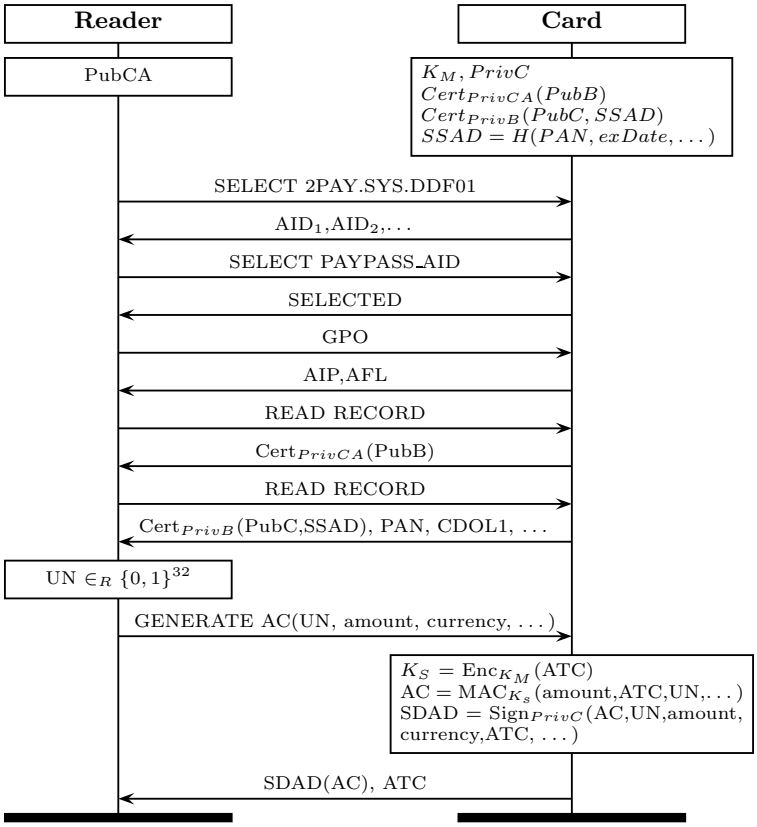
The original contact-based EMV standard consists of four books, containing over 700 pages [13] and the EMV contactless specification [14] builds on this. Additionally, every payment provider has their own variation of the payment protocol (Book C, Kernel Specification). Currently there are 7 different variations, the most widely used of which are from MasterCard (PayPass) and by Visa (payWave). The protocols are not directly compatible with each other (although they do use the same set of basic commands), therefore a shop reader must start a transaction by querying a card to see which protocols it supports.

The aim of the payment protocols is to provide evidence to the shop reader that the card is genuine and to give the shop reader a cryptogram (referred to as the AC), which it can send to the bank as proof that a payment is due. To achieve this cards have a public key pair (PubC,PrivC), and a certificate for the public key, signed by the bank, which includes all of the card's static information, $(Cert_{PrivB}(PubC, SSAD))$ the card also includes a certificate for the banks public key $(Cert_{PrivCA}(PubB))$ and the reader has the public key for this certificate (PubCA). To form the cryptogram the card also contains a symmetric key K_M, which is only known by the card and the bank.

The steps of Mastercard's PayPass protocol are shown in Figure 1. The first four steps initialise the protocol: the shop reader selects the payment application and the card replies with a list of application identities (AIDs) for the protocols it supports. The reader will then select the protocol it wishes to run. Next, the reader sends a "GET PROCESSING OPTIONS" (*GPO*) command and the card replies with the Application Interchange Profile (AIP), which indicates the specific functions the card supports, and the location of the records on the card (AFL) . The card will read these records, which include all the information printed on the card (except for the CVV, but including the credit card number, referred to as the PAN), the card's public key

[4] http://www.cs.bham.ac.uk/~tpc/Relay/

Fig. 1 The PayPass Protocol

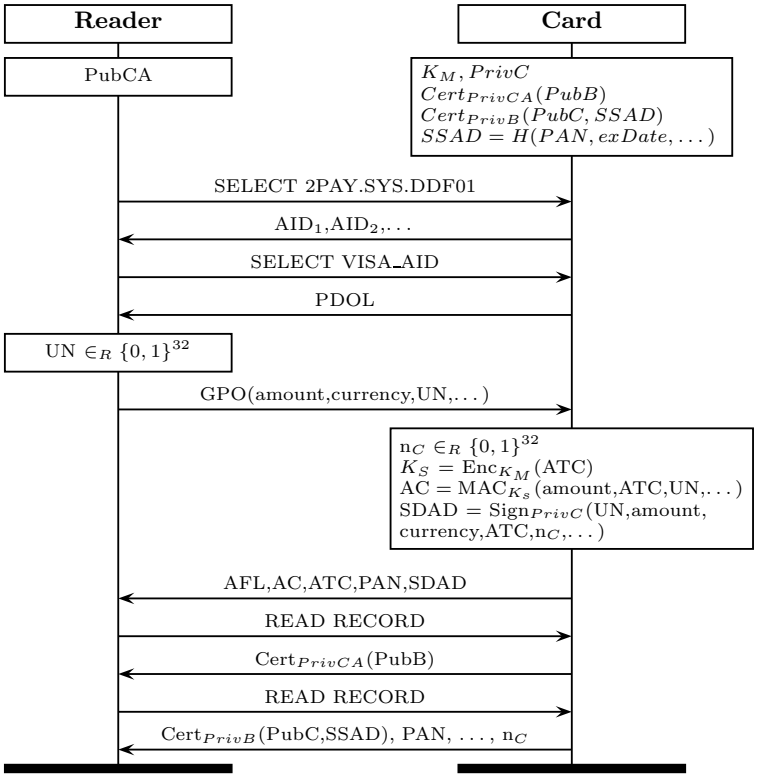


and the card's public key certificate. The reader also reads the Card Risk Management Data Object List (CDOL) which lists the information that the reader must provide to the card so that it can make the cryptogram for the bank. The number of read actions needed varies between cards and is specified by the AFL message.

Once the data has been read from the card, the reader generates a nonce UN, and requests the cryptogram with the GENERATE AC command. This command includes all of the information requested by the card, which will include the nonce, the amount of the transaction and the currency, among other things. The card then generates a session key (K_S), based on the key it shares with the bank and the application transaction counter (ATC), which equals the number of times the card has been used. The session key is then used to calculate a MAC on the transaction details (the AC), which is what the shop sends to the bank as proof of the transaction. With no access to the symmetric key the shop cannot check the AC, so to provide evidence to the shop that the card is genuine it signs AC along with the reader's nonce and the transaction details, which is referred to as the Signed Dynamic Application Data (SDAD).

The reader checks the card's public key certificate using the copy of the bank's verification key, uses this to check the SDAD, and if all of the data is correct the shop accepts the transaction. Complete transaction traces, with all information fields

Fig. 2 Visa's payWave qVSDC Protocol



parsed and explained are available on our website (apart from the credit card numbers which have been removed).

Visa's payWave qVSDC protocol is shown in Figure 2. This protocol is a slightly compressed version of Mastercard's PayPass protocol. The main differences are that the card provides the list of the information it requires (the PDOL) in response to the protocol being selected, and this information is provided to the card as part of the GPO message. The card will calculate the AC and sign the data in response to the GPO message. The GENERATE AC command is no longer used, because all of it's functionality is merged with the GPO message. After this command the terminal reads the files indicated in the AFL and can then authenticate the data after the card has left the field.

The data authentication mechanism (referred to as fDDA) generates a signature that includes a nonce from the card, and it is returned together with the cryptogram in the response to the GPO command, however, unlike PayPass, the signed data does not include the cryptogram. Therefore, it would be possible for an attacker to send a shop reader a valid SDAD and an invalid AC, which the shop would not discover until it send the AC to the bank for payment, this is known as the DDA wedge attack [19]. For a more detailed description of the EMV protocol we refer the reader to [9].

3 Relay Attacks Against EMV Contactless Smart Cards

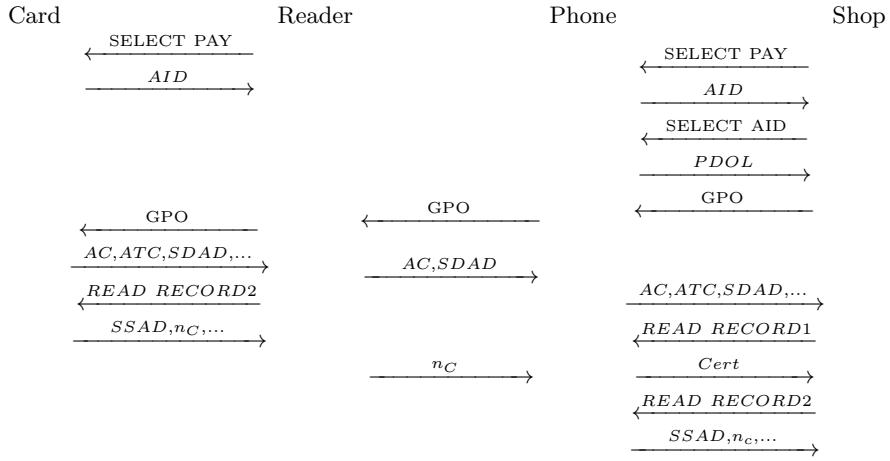
Existing implementations of Contactless EMV provide little or no protection against relay attacks. One of the most popular payment terminals in the UK imposes a limit on the total transaction time of 10 seconds. Such an ample time window allows an adversary to relay a transaction to everywhere in the planet (or even the moon).

A number of generic NFC relay attacks have been proposed in the literature [17,16]. Furthermore, successful relay attacks on payWave have been reported in, for example, [11]. Their average time for a relayed transaction is 1.6s, which is an overhead of about 1.1s compared to a regular transaction.

Our setup. We have conducted experiments with both MasterCard's PayPass and Visa's payWave, and were able to relay transactions with both systems. Our setup consists of an NFC mobile phone to emulate the bank card and we use another NFC phone or an off-the-shelf USB reader for the terminal side. Our implementation is multi-threaded and the data is relayed over Wifi. The relay is performed in two stages, first the reader or phone that is communicating with the card runs the protocol and records all of the static data, which is sent to the phone that will emulate the card. In the second phase, for payWave cards, the phone relays the cached data to the reader, while the card is simultaneously selected so that it is ready to reply to the relayed GPO message. The card is also asked for its nonce, which is relayed while the shop reader is reading the records, so this relay doesn't affect the overall time taken. Figure 3 gives an overview of the second stage of our payWave relay attack.

For PayPass we also cache the static messages and relay the GENERATE AC command, rather than the GPO. Additionally, we found that Android phones would put the Wifi adapter to sleep after about 100ms of inactivity, so to avoid this slowing down the relay we send a heartbeat signal over Wifi every 80ms. Our average transaction times for relayed payments are 623ms for PayPass and 550ms for payWave, much faster than times previously reported.

Fig. 3 The communications made by our payWave relay after pre-caching



Relay and timing. A first attempt to prevent such relay attacks would be to set an over all time bound on the protocol. The Contactless EMV specification [14] states that the complete transaction should take less than 500ms, however we found that shop readers did not enforce this. Furthermore, we could complete a relayed session with some cards faster than this; the fastest card was from the Dutch bank Knab and we could complete a relayed session with this card in 485ms. The slowest card we found was one from ABN-AMRO which took on average 637ms to complete a transaction (when not being relayed). This card was so slow that, thanks to the caching of static messages, our relay setup is able to complete a transaction with a shop reader using this card in 627ms, i.e., 10ms *faster* than when the card is directly communicating with the terminal. Hence, an overall time limit on the protocol cannot be used to stop relay attacks.

A second attempt to prevent relay attacks could be to set a (tighter) time bound on the dynamic messages of the protocol, i.e., the messages that need to be relayed. For PayPass this would be the GENERATE AC command, and for payWave the GPO command. These are also the commands that require the card to do some cryptographic computations which we have found leads to a large variance in the response times. For PayPass we observed average timings to GENERATE AC messages from 154ms to 332ms for different cards, where the overhead introduced by our relay for this message is only 161ms on average.

Another factor that increases the time variance for computationally intensive operations is the positioning of the card within the electromagnetic field of the reader. Experiments on a payWave card (from TSB bank) show that it takes on average 108ms to respond to the GPO command when the card is placed directly on top of the reader. Although, the same card takes on average 184ms to respond to the GPO command when the card is placed farther away from the reader (but still within range). The fastest response to the GPO message we received from a payWave card took 105ms, whereas the slowest took 364ms while the shortest time we observed when relaying this message was only 208ms. Hence, in some cases, the observed time variance between various cards was actually larger than the overhead introduced by the relay, so bounding the GPO and GENERATE AC message response times does not seem to be a practical way to stop relay attacks.

For the payWave protocol, an additional message needs to be relayed, namely the card nonce N_c. The cards do not need to perform any computation to reply to this message, and we found the time taken to reply was much more consistent. Experimenting with a number of different cards in different positions on the reader, we found that the fastest response took 20ms and the slowest took 68ms. So the time taken to relay this message would be noticeable to the shop terminal, however this message only contains dynamic data from the card and the command by the terminal is static, so this can be read and cached before it is requested by the shop reader (as our relay does). This means that it is also infeasible to detect relay attacks on this message using time bounds.

For our card timing experiments we used an Omnikey 5321 v2 USB reader. We note that different readers may produce different times, due to the power they provide to the card and the speed of the drivers, however the variances between the timings due to the card will be consistent. For instance, we found that the Nexus 5 phones we

used where typically 15ms to 30 ms slower than the USB reader. Another factor that affects the times is the length of the messages. For example, requesting the number of PIN tries is the shortest message that the card can produce and when asked for this the card takes approximately 8ms to reply and relaying such a short message (a couple of bytes) over Wifi takes at least 30ms.

Below, we propose a lightweight distance bounding protocol that prevents relay attacks using off-the-shelf devices such as mobile phones. Our protocol is able to prevent such attacks while tolerating the large computing-time variance of the cards. It is worth mentioning that our protocol does not attempt to stop powerful adversaries with custom made hardware (e.g. [15], which cost more than US$ 100K [7]). Considering that contactless payments are limited to small amounts, the cost of the hardware would be a disincentive for criminals. A video of our relay in action, and details of all our time measurements as well as a full trace of our relay are available on our website.

4 A Payment Protocol that Defends Against Relay Attacks

In the previous section, we saw that a relay based on mobile phones or USB NFC readers adds a delay to the response. This delay is small when compared to the variance of the time it takes for the card to reply to a request that requires encryption, but it is large, and detectable, when compared to the time it takes the card to reply to messages that do not require any computation. Below we will discuss a protocol that can be used to prevent relay attacks using mobile phones or USB NFC readers. This hardware is cheap and easy to get and using it in a store does not raise any suspicion, as companies such as Apple are introducing their own NFC apps to perform payments.

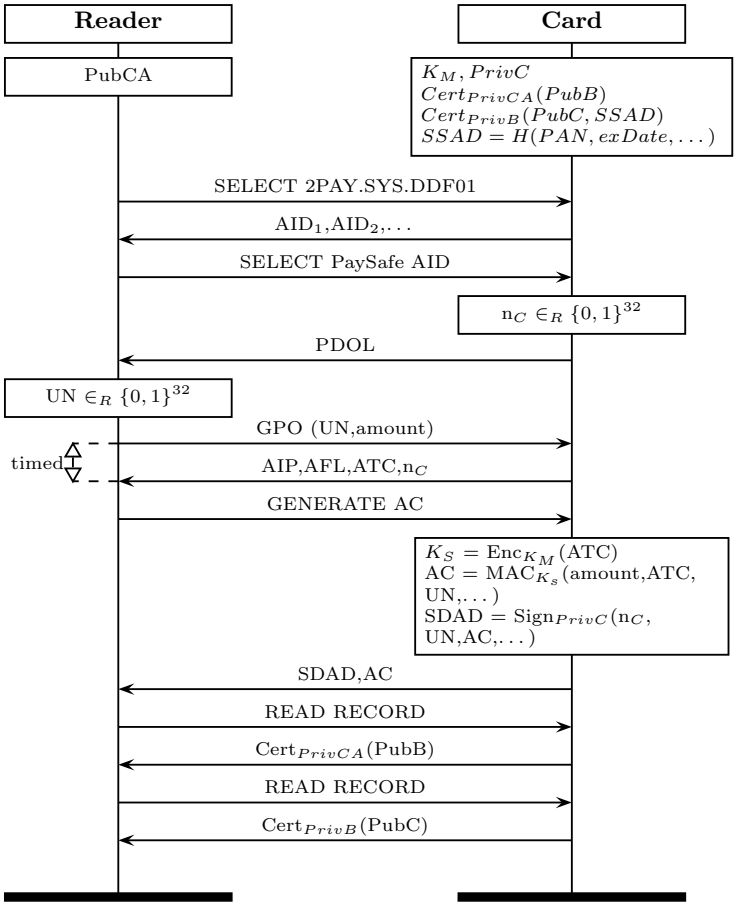
The problem with using the small delay caused by the relay to detect attacks is that this cannot reliably be used for commands that carry out cryptographic computations or return data that can be cached. The time of the cryptographic computations will vary due to for instance, the card's hardware or placement in the field, and so cannot be reliably bound. We fix this problem by splitting the challenge and response command from the generation of the signed authentication and cryptogram. To do this we provide the reader's nonce to the card with the GPO command (as in the payWave protocol), but we delay the generation of the signed authentication and cryptogram until the reader issues the GENERATE AC command (as in the PayPass protocol).

Our protocol, which we call the PaySafe protocol, is shown in Figure 4. In reply to the GPO command, the reader will provide its own nonce that was generated at a previous step in the protocol (i.e. when receiving the SELECT command). As the reply to the GPO command now does not require any computation, it can be timed by the reader to detect relay attacks. To detect an attempt by an attacker to defeat the timing bound by generating their own nonce to replace the reader's or card's nonce, both the reader's and the card's nonce are included in the signed data, as is the AC, so fixing VISA's problem of allowing an attacker to inject an invalid AC.

We use existing fields within EMV to exchange the nonces, namely the Unpredictable Number and the ICC Dynamic Number for the reader and card respectively. This means the cards are still EMV compliant and additionally, since both values are included in signed data, we do not need to add an additional signature to the protocol.

Our new proposed protocol will start in the same way as the existing EMV Contactless protocols with the selection of the payment application. After the application

Fig. 4 The protocol PaySafe which defends against simple relay attacks



is selected, and before the card sends its request for data to the reader (the PDOL) the card will generates a nonce. This nonce is not sent to the reader at this point but just stored on the card.

The next step is a timed GPO command, that gives the card the data it needs and the reader's nonce. The card will immediately reply to this message with the stored nonce. As this does not require any computation the card will reply quickly and without much variance in the time taken. If this message was relayed, additional overhead would be introduced by the network communication between the phones and the communication between a phone and the genuine terminal. The exact timing will depend on the hardware used; with our hardware the reply to the PayPass GPO message took 36ms and the PaySafe message is slightly longer. Therefore, we would suggest a time out of 80ms, as being long enough to make sure all cards are accepted, but still quick enough to make sure the message cannot be relayed using NFC phones.

To get the cryptogram and Signed Dynamic Application Data (SDAD) from the card, the reader issues the GENERATE AC command. The card then computes the

Fig. 5 Syntax of the Applied Pi Calculus

$M, N ::=$	terms
x, y, z	variables
a, b, c, k, s	names
$f(M_1, \ldots, M_n)$	constructor application
$D ::= g(M_1, \ldots, M_n)$	destructor application
$P, Q ::=$	processes
0	nil
$\overline{M}\langle N\rangle.P$	output
$M(x).P$	input
$P \mid Q$	parallel composition
$!P$	replication
$\nu\, a.P$	create new name
let $x = D$ in P else Q	term evaluation
$n : P$	phase

SDAD and the cryptogram and sends these to the reader. Attackers can get to this point in the protocol by sending their own nonce to either the card or the reader, so avoiding the need to relay both nonces and meeting the time restriction. To detect this both the reader's and the card's nonces are included in the signed data (SDAD). The reader will check these and terminate the transaction if they do not match the nonces that were part of the timed challenge.

After running this protocol the reader can be sure that fast nonce exchange took place with an entity that was not being relayed using mobile phones or USB NFC reader links, and, due to the SDAD, the reader can be sure that this entity was the bank card. Variations of this protocol are also possible, for instance the PDOL could be replaced with a CDOL in one of the records and the GENERATE AC message could be moved to the end, to make it much closer to the current PayPass protocol. It would also be possible to move the AIP,AFL and ATC data from the reply to the GPO into the reply to the SELECT AID, so making the reply to the GPO quicker and easier to time, however this should not be necessary and would be a further deviation from the current EMV protocols.

5 Formal Verification of PaySafe

The Applied Pi-calculus and ProVerif. To formally verify the correctness of our protocol we use the applied pi-calculus [2] and the tool ProVerif [4]. The applied pi-calculus is a formal language for describing protocols using the syntax given in Figure 5. This language allows us to specify processes that perform inputs and outputs, run in parallel and replicate. The calculus also allows processes to declare new, private names which can be used as private channels or nonces [4]. Functions in the applied pi-calculus can be used to model a range of cryptographic primitives, e.g MACs, signing and key generation. The "let" statement can be used to check that two terms that used these equations are equal and branch on the result. This can be used to encode "if" statements, and conditional inputs $c(=\mathrm{a}).P$ which inputs a value and proceeds only if the value received equals a (see e.g. [4]). ProVerif is an automatic theorem proving tool that can be used to check applied pi-calculus models. It can be used to

check secrecy of, for example, a key or, as we do below, it can be used to check if a process can reach a certain point. ProVerif introduces phase statements, written as $n:P$ where n is an integer. Processes will be run in the order of the numbers they are tagged with, which enforces an ordering on the commands, e.g. an output in phase 2 cannot be received by an input in phase 1.

Modelling PaySafe. The applied pi-calculus model of our new protocol is given in Figure 6. The first two communications between the card and the reader are the SELECT commands. The card listens first for the selection of the payment environment and then the selection of the particular payment application, but before replying to the second SELECT command it declares a new name n_C. The reader also generates a new name n_R and passes that to the card using the GPO command and requests the cryptogram, by sending the GENERATE AC command. The encryption is done using "let" statements and the reader checks the card's signature using an "if" statement before accepting it. The *System* process includes an arbitrary number of readers and cards, sets the card information and makes the bank's public key available to the attacker.

Verifying the Defence Against Relay Attacks. Based on our observations in Section 3, an attacker cannot use a mobile phone or USB equipment to relay the GPO message in our new protocol quickly enough to make the reader accept the reply. However, this alone is not enough to guarantee that our protocol stops relay attacks; it may still be possible for attackers to pre-play messages to the card, or make up their own response to the GPO message and trick the reader into believing that it came from the card.

As the attackers cannot relay the timed message to the card and get the answer back to the reader quickly enough, the attackers cannot have a meaningful interaction with the card while the reader is waiting for the reply. I.e., the attackers can perform a relay attack if, and only if, they can do so without communicating with the card between when the reader broadcasts its timed message and when it receives a reply.

Our formal modelling of relay attacks is based on this observation. Between when the reader broadcasts its timed message and when it receives the reply we lock the card process so that it cannot communicate with anyone. We also lock the other readers in the system as the attacker will not have time to relay messages to them, as well as the card. Given these locks, if the attacker can find a sequence of actions that still allows a reader to finish the protocol then a relay attack is possible. On the other hand, if the locks stop the reader from terminating then there is no sequence of relayed actions that the reader will accept from the attacker, and we can be sure that the protocol is safe from relay attacks.

We could encode a lockable process in the applied pi-calculus by adding a single 'heartbeat' output on a private name. The lockable process must acquire this before performing any input or output, and releases it again afterwards. To lock this process another process only needs to acquire the lock. A model like this could be checked by hand in the applied pi-calculus, and so provides a useful analysis method, however it cannot be checked automatically using the ProVerif tool. ProVerif can prove correctness for a protocol with an arbitrary number of runs, and an arbitrary number of concurrent processes. To make this possible ProVerif makes some compromises: the

Fig. 6 Applied pi-calculus model of PaySafe

$$
\begin{array}{ll}
\mathit{Reader} = \overline{c}\langle\text{SELECT,PAYSYSDDF}\rangle. & \mathit{Card} = c(=\text{SELECT},=\text{PAYSYSDDF}).\\
\quad c(=\text{AID}). & \quad \overline{c}\langle\text{AID}\rangle.\\
\quad \overline{c}\langle\text{SELECT,aid}\rangle. & \quad c(=\text{SELECT},=\text{AID}).\\
\quad c(=\text{PDOL}). & \quad \nu\, n_C.\overline{c}\langle\text{PDOL}\rangle.\\
\quad \nu\, n_R.\overline{c}\langle\text{GPO,amt},n_R\rangle. & \quad c(=\text{GPO,amt}',n_R').\\
\quad c(n_C',\text{atc}',\text{pan}'). & \quad \overline{c}\langle n_C,\text{atc,pan}\rangle.\\
\quad \overline{c}\langle\text{GENERATE AC}\rangle. & \quad c(=\text{GENERATE AC}).\\
\quad c(\text{sdad}',\text{ac}'). & \quad \text{let mac}_K = \mathit{genkey}(\text{atc,bank}_K) \text{ in}\\
\quad \overline{c}\langle\text{READ RECORD}\rangle.c(\mathit{ssad}'). & \quad \text{let ac} = \mathit{mac}((\text{amt}',n_R',\text{atc}),\text{mac}_K) \text{ in}\\
\quad \overline{c}\langle\text{READ RECORD}\rangle.c(\text{cert}'). & \quad \text{let sdad} =\\
\quad \text{let cardPub}_K' = & \qquad \mathit{sign}((n_R,n_C,\text{amt,atc,ac}),\text{card}_K) \text{ in}\\
\qquad \mathit{check}(\text{cert}',\mathit{pk}(\text{bank}_K)) & \quad \overline{c}\langle\mathit{sdad}\rangle.\\
\quad \text{if } \mathit{check}(\text{sdad}',\text{cardPub}_K') = & \quad c(=\text{READ RECORD}).\\
\qquad (n_R,n_C',\text{amt,atc}',\text{ac}') & \quad \overline{c}\langle\mathit{sign}((\text{pan,expDate}),\text{bank}_K)\rangle.\\
\quad \overline{c}\langle\text{readerAccepts}\rangle & \quad c(=\text{READ RECORD}).\overline{c}\langle\mathit{cert}\rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{System} = \ \nu\,\text{bank}_K.(\overline{c}\langle\mathit{pk}(\text{bank}_K)\rangle \mid\ !\nu\,\text{amount}.!\mathit{Reader}\\
\quad \mid\ !(\nu\,\text{pan}.\nu\,\text{expDate}.\nu\,\text{card}_K.\text{let cert} = \mathit{sign}(\mathit{pk}(\text{card}_K),\text{bank}_K) \text{ in } !\nu\,\text{atc}.!\mathit{Card}))
\end{array}
$$

tool may not terminate and it may report false attacks. Using a heartbeat lock, as described before, results in ProVerif finding a false attack because the tool wrongly allows more than one process to acquire the lock at the same time.

Encoding a Process Lock Using Phases. To create a model that can be checked in ProVerif, and so automatically prove our protocol correct, we can use ProVerif's phase statements. To model relay attacks we use three phases $(0, 1, 2)$ and we allow the card and reader to be interacted with in phases 0 and 2. The attacker can act in all of the phases and the reader will broadcasts its request for the timed message and receive the reply in phase 1. This stops the attacker from relaying the timed message to the card and forwarding the response to the reader, but it still allows the attacker to try to replay or pre-play messages to the card, and make up its own responses to the challenge.

We make the assumption that there may be concurrent sessions of cards and readers (although the readers and cards we looked at didn't support this, future devices may). Therefore, we need to allow the reader and card process to span phases 0 and 2, but not act in phase 1, i.e. the processes must be able to jump from phase 0 to phase 2 before input actions (output actions can be forward from one phase to the next by the attacker). For the process P the following function gives us the set of processes produced by placing a phase 2 jump in front of each of the possible inputs:

$$\mathit{phasesSet}(P) = \{C[2{:}M(x).P'] \mid P = C[M(x).P']\}$$

Here $C[\_]$ ranges over all possible contexts, so the right hand side of this set comprehension matches all inputs in P, and the left hand side adds a phase command in front of the input.

We can then build a process that allows the attacker an arbitrary number of interactions with the process P, in either phase, by replicating each member of this

Fig. 7 Our Protocol modelled using ProVerif's phase statements

$$
\begin{array}{ll}
\mathit{TestReader} = \ldots & \mathit{SystemP} = \nu\,\mathrm{bank}_K.(\overline{c}\langle pk(\mathrm{bank}_K)\rangle \\
\quad c(=\mathrm{PDOL}).\nu\, n_R. & \quad\quad \mid \nu\,\mathrm{amount}.\mathit{TestReader} \\
\quad 1{:}\overline{c}\langle \mathrm{GPO}, \mathrm{amt}, n_R\rangle. & \quad\quad \mid\ !\nu\,\mathrm{amount}.\mathit{Readers} \\
\quad c(n'_C, \mathrm{atc}', \mathrm{pan}'). & \quad\quad \mid\ !(\nu\,\mathrm{pan}.\nu\,\mathrm{expDate}.\nu\,\mathrm{card}_K. \\
\quad 2{:}\overline{c}\langle \text{GENERATE AC}\rangle. & \quad\quad\quad \text{let cert} = \mathit{sign}(pk(\mathrm{card}_K), \mathrm{bank}_K) \\
\quad c(\mathrm{sdad}', \mathrm{ac}'). & \quad\quad\quad \text{in } !\nu\,\mathrm{atc}.\mathit{Cards})) \\
\quad \ldots & \\
\quad \text{if } \mathit{check}(\mathrm{scad}', \mathrm{cardPub}'_K) = & \text{where:} \\
\quad\quad (n_R, n'_C, \mathrm{amt}, \mathrm{atc}', \mathrm{ac}') & \quad \mathit{Cards} \quad = \mathit{phases}(\mathit{Card}) \\
\quad \overline{c}\langle \mathrm{phaseReaderAccepts}\rangle & \quad \mathit{Readers} = \mathit{phases}(\mathit{Reader})
\end{array}
$$

set and placing them in parallel:

$$\mathit{phases}(P) = !P_1 \mid !P_2 \mid \cdots \mid !P_n \quad \textit{where} \quad \{P_1, \ldots, P_n\} = \mathit{phasesSet}(P)$$

We also note that if the attacker can perform a relay attack against our system process, then there is a single distinct reader process that the attacker can get to terminate, while relaying messages. This means that it is sufficient to test if just a single reader process can terminate, and only this reader process needs to enforce the timing restrictions.

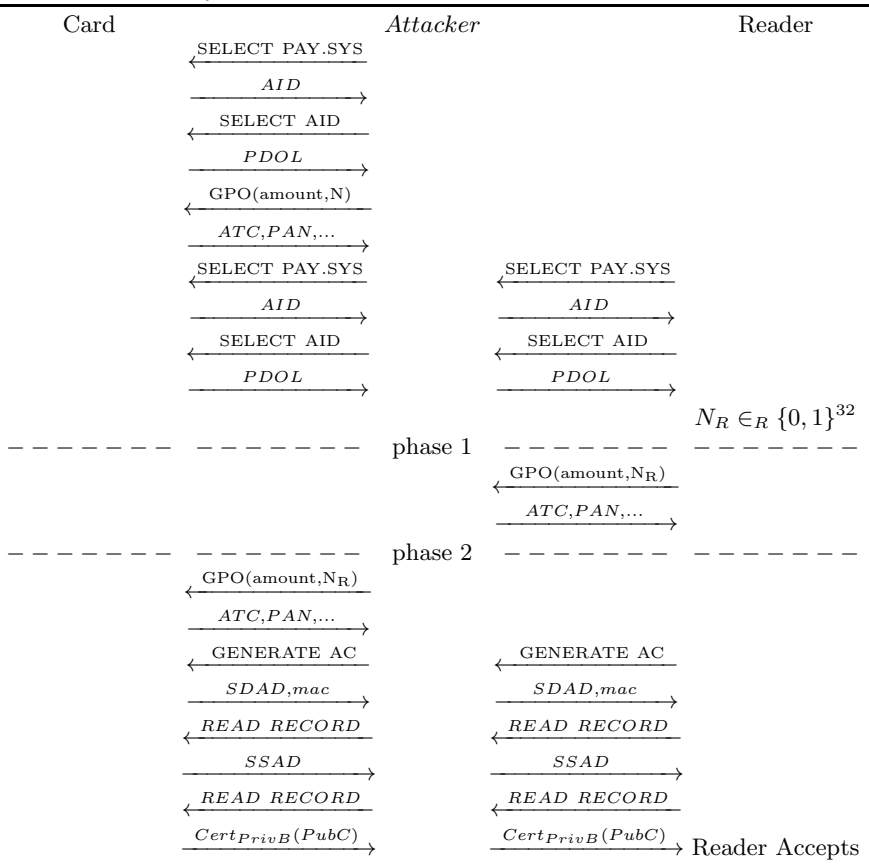
We use the *phases* function to give the attacker an arbitrary number of copies of the card and the reader they can interact with in both phases 0 and 2, and we add a single copy of the reader process that outputs its GPO message and must receive the timed relay in phase 1. This is shown in Figure 7, where *Reader* and *Card* are as defined in Figure 6.

Checking the *System* process with ProVerif, we find that the reader process cannot reach the *readerAccepts* action. Therefore, given the timing assumptions, we may conclude that the attacker cannot cause the reader to accept a session of the protocol, and that our protocol will stop relay attacks using cheap hardware, such as mobile phone and USB NFC readers.

Examples of Finding Attacks. To test our framework's ability to find attacks we look at what happens if the readers or the cards did not use a nonce. First we removed the card's nonce from the protocol, both in the reply to the GPO message and in the check made by the reader, however we keep the timing constraints on the GPO message as imposed by the phase statements. When given this protocol model, ProVerif finds the attack illustrated in Figure 8. In this attack, the attacker first interacts with the card to learn the current counter value and the PAN. After this, the attacker relays messages from the reader, until the reader issues its GPO message which the attacker can now reply to directly, so meeting the timing constraint. The attacker then forwards these messages to the card and continues to relay the remaining messages from the reader.

The points at which the ProVerif model changes phase are also shown in Figure 8, and this illustrates how our method locks the card during the readers timed message. We also tested our protocol with a card nonce, but without the reader's nonce; in this case ProVerif finds a simple replay attack: the attacker interacts with the card and records a whole session, which can then be replayed to the reader's GPO message

Fig. 8 Attack found by ProVerif if there is no card nonce



directly, so meeting the timing requirements. All of our ProVerif models are available on our website.

6 Implementing PaySafe

The protocol we propose has the advantage of only using existing EMV commands and data fields. As it is similar to the existing contactless EMV protocols, it is simple to implement. The changes from the payWave qVDSC protocol are the removal of the Signed Dynamic Application Data from the response to the GPO message, moving the card's nonce data field into the response to the GPO message and the addition of the standard EMV GENERATE AC command. The changes from the PayPass protocol are even smaller, requiring only the moving of the reader's nonce data field to the GPO message and the card's nonce data field into the response to the GPO message.

The EMV framework allows for multiple applications to be offered by a card. So our new protocol could be assigned a new AID and if both reader and card support the protocol it would be used. The AIDs supported will be included in the signed card information, so attackers cannot trick a reader into using a less secure protocol.

Care must be taken to ensure that deviations from the specified protocol do not make attacks possible. For instance, if cards would accept a second GPO message with a different reader nonce, an attacker could discover the card nonce with the first GPO message and then relay the readers nonce with the second GPO message. So, for PaySafe we must stop a single run of the protocol from accepting multiple GPO messages with different nonces. Clearly, our protocol only works if the nonces are actually unpredictable, unfortunately some EMV terminals have been shown to use predictable numbers [5].

7 Conclusion

Our goal in this paper was to propose a defence against relay attacks using easily available hardware on the EMV Contactless protocol, and to prove it correct. Our solution relies on the quick exchange of (previously generated) random nonces. This solution fits within the EMV Contactless specification, but is applicable to smart card protocols in general. We also have presented a new technique to model relay attacks based on a simple observation: if the reply to the timed message is received by the reader quickly enough, then the attacker cannot have interacted with the card. We have shown how this can be used to model relay attacks in the applied pi-calculus and in the automatic tool ProVerif using phase statements. As further work, we could extend our formal models to include all error handling for the protocols, as well as developing our method of modelling relay attacks into a general framework for any protocol.

Acknowledgement

We would like to thank Chris Smith, Ben Smyth, Alexander Darer, Mandeep Daroch and a number of helpful shop staff for their assistance with developing the relay.

References

1. Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *J. ACM*, 52(1):102–146, January 2005.
2. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Symposium on Principles of Programming Languages (POPL)*, 2001.
3. Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE, 2001.
4. Bruno Blanchet, Ben Smyth, and Vincent Cheval. ProVerif 1.88: Automatic cryptographic protocol verifier, user manual and tutorial, 2013.
5. Mike Bond, Omar Choudary, Steven J. Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and Skim: cloning EMV cards with the pre-play attack. In *35th IEEE Symposium on Security and Privacy*, 2014.
6. Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Towards secure distance bounding. In *Fast Software Encryption - 20th International Workshop, FSE 2013*, pages 55–67, 2013.
7. Srdjan Capkun. Personal communication, 2012.
8. Cas Cremers, Kasper Bonne Rasmussen, Benedikt Schmidt, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *2012 IEEE Symposium on Security and Privacy (SP)*, pages 113–127. IEEE, 2012.
9. Joeri de Ruiter and Erik Poll. Formal analysis of the EMV protocol suite. In S. Mödersheim and C. Palamidessi, editors, *Theory of Security and Applications*, volume 6993 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2012.

10. Saar Drimer and Steven J. Murdoch. Keep your enemies close: distance bounding against smartcard relay attacks. In *USENIX Security Symposium*, pages 87–102, August 2007.
11. M. Emms, B. Arief, T. Defty, J. Hannon, F. Hao, and A. van Moorsel. The dangers of verify PIN on contactless cards. Technical report. CS-TR-1332.
12. Martin Emms, Budi Arief, Leo Freitas, Joseph Hannon, and Aad van Moorsel. Harvesting high value foreign currency transactions from emv contactless credit cards without the pin. In *21st Conference on Computer and Communications Security (CCS)*, 2014.
13. EMVCo. EMV – Integrated Circuit Card Specifications for Payment Systems, version 4.3, 2011.
14. EMVCo. EMV Contactless Specifications for Payment Systems, version 2.4, 2014.
15. Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011*. The Internet Society, 2011.
16. Lishoy Francis, Gerhard Hancke, and Keith Mayes. A practical generic relay attack on contactless transactions by using NFC mobile phones. *International Journal of RFID Security and Cryprography (IJRFIDSC)*, 2(1-4):92–106, 2013.
17. Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical NFC peer-to-peer relay attack using mobile phones. In *Proceedings of the 6th International Conference on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec'10, pages 35–49. Springer, 2010.
18. Gerhard Hancke and Markus Kuhn. An RFID distance bounding protocol. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 67–73. IEEE, 2005.
19. Steven J. Murdoch. Defending against wedge attacks in Chip & PIN. URL: https://www.lightbluetouchpaper.org/2009/08/25/defending-against-wedge-attacks/.
20. Luigi Sportiello and Andrea Ciardulli. Long distance relay attack. In Michael Hutter and Jörn-Marc Schmidt, editors, *Radio Frequency Identification*, Lecture Notes in Computer Science, pages 69–85. Springer, 2013.

A Alternative Versions of the PaySafe Protocol

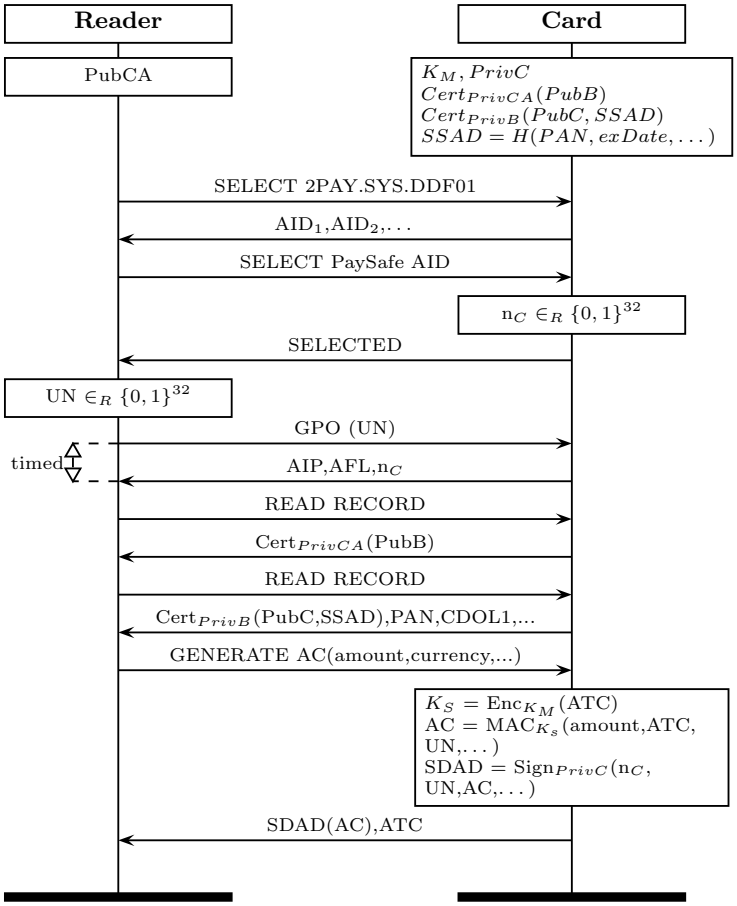
The version of the protocol given in the main paper is closest to Visa's PayWave protocol. However the messages of the protocol could be reordered to make it more like Mastercard's PayPass protocols, as shown in Figure 9. In this version of the protocol, the only differences from PayPass are moving the readers unpredictable number from the GENERATE AC message into the GPO message, and adding a card nonce in the reply to the GPO message.

The time it takes to send messages is dependent on the length of the messages. To make the timed GPO message even harder to relay we could make the response to the GPO shorter by moving the AIP and AFL data into the response to the SELECT message, as shown in Figure 10. This will make the time taken by the relay even easier to spot, as a proportion of the time the message takes. However, our timing experiments show that this change is not necessary, and it has the disadvantage of making the protocol more different from the existing payment protocols.

B Relayed payWave transaction trace

This section presents an example trace from our relay, when relaying the payWave qVDSC protocol, with timings and the hex of each message. We also include a break

Fig. 9 A version of PaySafe which is closer the PayPass

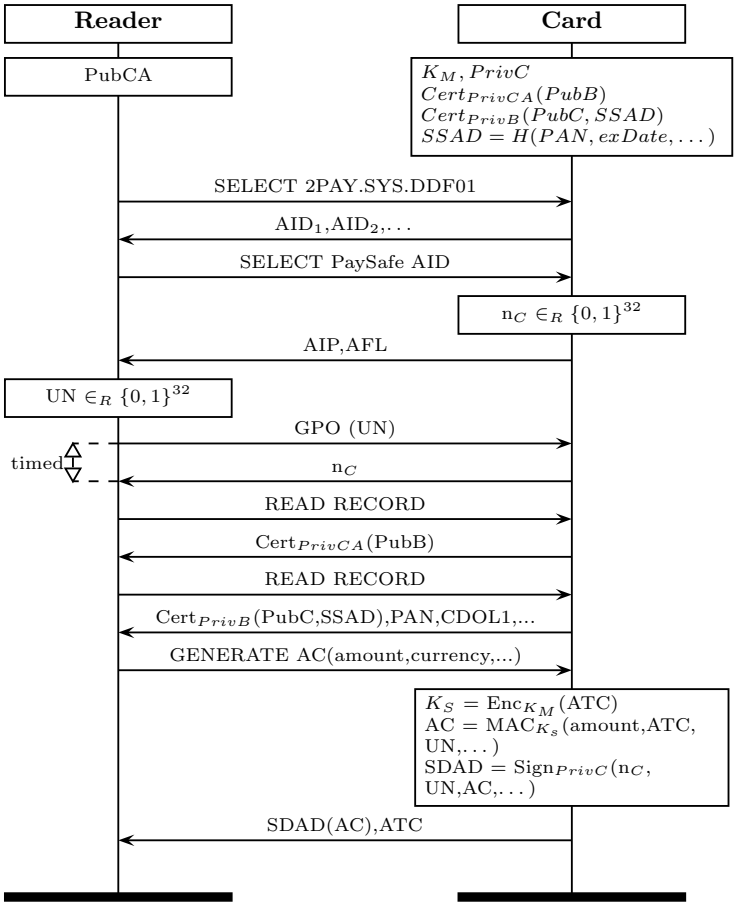


down of each message explaining its meaning. Each of the digits of the credit card number of the card used has been replaced with X.

```
Run time: 548ms total, 247ms to send GPO and get the answer back

2014-09-15 14:10:05.394 | NetworkTask  | Sending SELECT message to wake up
card.
2014-09-15 14:10:05.476 | MainActivity | Card responds:
6F378407A0000000031010A52C500A5649534120444542495 49F38189F66049F02069F03069F
1A0295055F2A029A039C019F37045F2D02656E9000
which parses as:
  6F | len:37    File Control Information (FCI) Template
      84 | len:7    DF Name: A0000000031010
      A5 | len:2C    Proprietary Information
          50 | len:10    Application Label: 56495341204445424954
        9F38 | len:18    Processing Options Data Object List (PDOL)
```

Fig. 10 A version of PaySafe which is closer the PayPass




```
          9F66 | len:04   Card Production Life Cycle
          9F02 | len:06   Amount, Authorised (Numeric)
          9F03 | len:06   Amount, Other (Numeric)
          9F1A | len:02   Terminal Country Code
            95 | len:05   Terminal Verification Results
          5F2A | len:02   Transaction Currency Code
            9A | len:03   Transaction Date00A4040007A000000003101000
            9C | len:01   Transaction Type
          9F37 | len:04   Unpredictable Number
       5F2D | len:2    Language Preference: 656E

--First message received from terminal at 14:10:08.186
2014-09-15 14:10:08.186 | PaymentService | Hex received from reader:
00A404000E325041592E5359532E444446303100
which parses as: SELECT PAY.SYS.DDF01
```

```
2014-09-15 14:10:08.191 | EmulatorApplet | Sending cache response to reader:
6F20840E325041592E5359532E4444463031A50EBF0C0B61094F07A00000000310109000
which parses as:
  6F | len:20    File Control Information (FCI) Template
      84 | len:14    DF Name: 325041592E5359532E4444463031
      A5 | len:0E    Proprietary Information
        BF0C | len:0B    File Control Information (FCI) Issuer Discretionary
         Data
              61 | len:09    Directory Entry
              4F | len:7    Application Identifier (AID): A0000000031010

2014-09-15 14:10:08.210 | PaymentService | Hex received from reader:
00A4040007A000000003101000
which parses as: SELECT A0000000031010

2014-09-15 14:10:08.217 | EmulatorApplet | Sending cache response to reader:
6F378407A0000000031010A52C500A5649534120444542495 49F38189F66049F02069F03069F
1A0295055F2A029A039C019F37045F2D02656E9000
which parses as above.

2014-09-15 14:10:08.252 | PaymentService | Hex received from reader :
80A800023832132204000000000000030000000000000082600000000000826140915003 38F
507800
which parses as:
    Card Production Life Cycle     32-20-40-00
    Amount, Authorised (Numeric)   000000000030
    Amount, Other (Numeric)        000000000000
    Terminal Country Code          0826
    Terminal Verification Results  0000000000
    Transaction Currency Code      0826
    Transaction Date               140915
    Transaction Type               00
    Unpredictable Number           338F5078

2014-09-15 14:10:08.255 | EmulatorApplet | Sending GPO command over relay
2014-09-15 14:10:08.498 | NetworkTaskTCP | len : 199
2014-09-15 14:10:08.506 | NetworkTaskTCP | Total message send time = 247ms
2014-09-15 14:10:08.556 | EmulatorApplet | Returning response from card :
7781C29F4B81804D8EC3F85EB28D9C8828E2238BFE8F922F89D08DEDA061DE7270CF6EB01510
9D58DC58B34706CED0BFA24A28ED3E6AE0B2908617D34199B0A3BD298187376F639F65203C84
EEE7BC60B4D14F649E67C62162CAF53045E8D5A2A99E39589483A28DF24941C6AF486FEEBA0A
8C6DB33978309EFF87FFF9984C9DECDFDCE6728DB19404100203009F1007060A0A0390000057
13XXXXXXXXXXXXXXXXD160420151400000001001F820220009F360200579F26083501E6BD0985
62889F6C0210009000
which parses as:
```

```
  77 | len:81    Response Message Template Format 2
    9F4B | len:128    Signed Dynamic Application Data (SDAD): 4D8EC3F85EB28D
    9C8828E2238BFE8F922F89D08DEDA061DE7270CF6EB015109D58DC58B34706CED0BFA24A
    28ED3E6AE0B2908617D34199B0A3BD298187376F639F65203C84EEE7BC60B4D14F649E67
    C62162CAF53045E8D5A2A99E39589483A28DF24941C6AF486FEEBA0A8C6DB33978309EFF
    87FFF9984C9DECDFDCE6728DB1
      94 | len:4    Application File Locator: 10020300
    9F10 | len:7    Issuer Application Data (IAD): 060A0A03900000
      57 | len:19    Track 2 Equivalent Data: XXXXXXXXXXXXXXXXD1604201514000
      0001001F
      82 | len:2    Application Interchange Profile: 2000
    9F36 | len:2    Application Transaction Counter: 0057
    9F26 | len:8    Application Cryptogram: 3501E6BD09856288
    9F6C | len:2    Card Transaction Qualifiers (CTQ): 1000

2014-09-15 14:10:08.526 | EmulatorApplet | Reading card nonce on a separate
thread, and storing it for later.
2014-09-15 14:10:08.541 | EmulatorApplet | Received card nonce : 9E90BD37


2014-09-15 14:10:08.625 | PaymentService | Hex received from reader : 00B202
1400
which parses as: READ RECORD 0214

2014-09-15 14:10:08.635 | EmulatorApplet | Sending cached resp. to terminal:
2014-09-15 14:10:08.661 | EmulatorApplet | Returning response from the cache:
7081C08F01079081906103C2C80BF24F3E7117BCF60F29B2DF92EFBF52E4AAA6F27FC0A13BE2
2EE638806C77F08FDC1F80717DC79D9B0D1F1FC137F648E537DC2A36DAD9B3F367189F35B958
3AFE400B6D72D2362F31C3AD3510FB20C890FFC2C8B43B2389E36877C6C6E4C7BFD7D3BABB12
04CF440EC9E9FB6D855EFEF4C6E105A0F46A732CAD8644924AED8A97614E36EED4C0EE19BBB2
54922488C9DC1A7E668C6867F5937F2A69729D1120E3D2954DBBC4C935A10A06857FB8F63E72
CB9F3201039000
which parses as:
  70 | len:81    Record Template
      8F | len:1    Certification Authority Public Key Index: 07
      90 | len:144  Issuer Public Key Certificate: 6103C2C80BF24F3E7117BCF60
      F29B2DF92EFBF52E4AAA6F27FC0A13BE22EE638806C77F08FDC1F80717DC79D9B0D1F1
      FC137F648E537DC2A36DAD9B3F367189F35B9583AFE400B6D72D2362F31C3AD3510FB2
      0C890FFC2C8B43B2389E36877C6C6E4C7BFD7D3BABB1204CF440EC9E9FB6D855EFEF4C
      6E105A0F46A732CAD8644924AED8A97614E36EED4C0EE19BBB254
      92 | len:36   Issuer Public Key Remainder: 88C9DC1A7E668C6867F5937F2A6
      9729D1120E3D2954DBBC4C935A10A06857FB8F63E72CB
    9F32 | len:1    Issuer Public Key Exponent: 03

2014-09-15 14:10:08.706 | PaymentService | Msg from reader : 00B2031400
which parses as: READ RECORD 0314
```

```
2014-09-15 14:10:08.715 | EmulatorApplet | Sending cached resp. to terminal:
2014-09-15 14:10:08.730 | EmulatorApplet | Returning response from card :
7081D19F468190B04DCF375EB59A6F941757E8BBC926C95EA748381A6F0CCCFEF7415A76CCB3
D424F801AFA5A3624D4C6694BD2281A70CED813AF297D72374F3CE2D8343FC9B3DDEBEA1A553
B0D2BA896CA134785334F61ACF3CA6EB10061A694B97A281C50C11A5D242AD492118D644C85E
AB2F44AC445659282A4AAB132E84A3C52F5D9AC9F6EE8E57CE8C7595A3C71FF422DEED59049F
4701039F481A75F342106927017D05443C3F53B310F08A69C382B72B3ACEC1DB5A08XXXXXXXX
XXXXXXXX5F3401005F24031604309F6905019E90BD379000
which parses as:
  70 | len:81    Record Template
    9F46 | len:144    ICC Public Key Cert: B04DCF375EB59A6F941757E8BBC926C95
    EA748381A6F0CCCFEF7415A76CCB3D424F801AFA5A3624D4C6694BD2281A70CED813AF29
    7D72374F3CE2D8343FC9B3DDEBEA1A553B0D2BA896CA134785334F61ACF3CA6EB10061A6
    94B97A281C50C11A5D242AD492118D644C85EAB2F44AC445659282A4AAB132E84A3C52F5
    D9AC9F6EE8E57CE8C7595A3C71FF422DEED5904
    9F47 | len:1    ICC Public Key Expo: 03
    9F48 | len:26   ICC Public Key Remainder: 75F342106927017D05443C3F53B310
    F08A69C382B72B3ACEC1DB
      5A | len:8  Application Primary Account Number (PAN): XXXXXXXXXXXXXXXX
    5F34 | len:1  (PAN) Sequence Number: 00
    5F24 | len:3  Application Expiration Date YYMMDD: 160430
    9F69 | len:5  Card Authentication Related Data: 019E90BD37

-- Final message sent back at 14:10:08.730
```

C ProVerif Protocol Model

This section contains the ProVerif model of our protocol which was used to verify its correctness.

```
(* Equations for Signatures, MACs and session key generation *)
fun mac/2.
fun sign/2.
reduc checksign(sign(x,y),pk(y)) = x.
reduc getmess(sign(x,y)) = x.
fun genSkey/2.

free c.
private free bankSignKey.
private free bankMacKey.
private free bankChannel.

data SELECT/0.     data PAYSYSDDF/0.    data AID/0.          data GPO/0.
data PDOL/0.       data READRECORD/0.   data GENERATEAC/0.

(* The reader can't accept a cryptogram without a real card being present *)
query evinj:readerAccept(cGram) ==> evinj:cardGen(cGram).
```

```
let Card =
  in (c,(=SELECT,=PAYSYSDDF)); out (c,AID);
  in (c,(=SELECT,=AID));new nonceCard; out (c,PDOL);
  in (c,(=GPO,amountToPay,nonceReader)); out (c,(nonceCard,atc,ccNo));
  in (c,=GENERATEAC); let macSessionKey = genSkey(atc,bankMacKey) in
  let transCryptoGram = mac((amountToPay,nonceReader,atc),macSessionKey) in
  event cardGen(transCryptoGram);
  let sdad = sign((nonceReader,nonceCard,amountToPay,atc,transCryptoGram),cardKey) in
  out (c, (sdad,transCryptoGram) );
  in (c,=READRECORD); out (c,cert).

let Reader =
  out (c, (SELECT,PAYSYSDDF)); in (c, appId);
  out (c, (SELECT,appId)); in (c,dataRequested); new nonceR;
  out (c,(GPO,amount,nonceR));in (c, (nonceC,counter,pan));
  out (c, GENERATEAC); in (c, (sig,cryptogram));
  out (c,READRECORD); in (c,cardCert);
  let cardKey = checksign(cardCert,pk(bankSignKey)) in
  if checksign(sig,cardKey) = (nonceR,nonceC,amount,counter,cryptogram) then
      out(bankChannel,(cryptogram,amount,ccNo));
  event readerAccept(cryptogram).

process ( !new amount;!Reader
  | !(new ccNo;new cardKey; let cert = sign(pk(cardKey),bankSignKey) in !new atc;Card)
  | out (c, pk(bankSignKey)) )
```