Designing and proving an EMV-compliant payment protocol for mobile devices

Véronique Cortier, Alicia Filipiak, Saïd Gharout, Jacques Traoré

Designing and proving an EMV-compliant payment protocol for mobile devices

Véronique Cortier
LORIA - CNRS

Alicia Filipiak
LORIA - Orange Labs

Saïd Gharout
Orange Labs

Jacques Traoré
Orange Labs

***Abstract*—We devise a payment protocol that can be securely used on mobile devices, even infected by malicious applications. Our protocol only requires a light use of Secure Elements, which significantly simplify certification procedures and protocol maintenance. It is also fully compatible with the EMV SDA protocol and allows off-line payments for the users.**

We provide a formal model and full security proofs of our protocol using the TAMARIN prover.

1. Introduction

Mobile devices have become life-style devices containing many user applications and allow access to several remote services. According to GSMA, the global Mobile Network Operators organisation, two thirds of world's mobile phones will be smartphones by 2020 [19]. This evolution of mobile devices allowed the development of many mobile applications and services (e.g. payment, transport, banking transactions, etc.). Today, with the emergence of open environments in mobile devices (e.g., Android), purposes of mobile applications have incredibly widened, hence everybody can now develop their own mobile applications and install it. However, such a success comes at a price. The great diversity of applications on mobile devices is one of the causes that raised frauds and attacks [20]. Mobile devices are prone to several attacks, such as gathering private information from the device like cryptographic keys, banking information or user's data. Hackers could for example fraudulently use the different applications and services proposed by these platforms, or install malware and try the usurpation of a legitimate application.

Of course, secure protocols have already been developed for years, even decades. However, standard protocols (eg. payment protocols) assume the host machines to be secure. For example, EMV [11], which has been used since 1994, is today the standard in payment industry and widely deployed by major payment networks (e.g., Visa, Mastercard, etc.). It has already been intensively studied (see e.g. [29], [10], [6], [24]) and is reasonably secure. *However, its security strongly relies on the keys owned by the user's credit card.* While it is very difficult to extract keys from credit cards, this is no longer the case for applications stored and run on mobile devices. A payment protocol should be secure even if *it runs on a device that offers low level of security* or contains malicious applications, to fight against frauds and cloning.

Several payment solutions have been developed such as Google Wallet, Apple Pay, or Orange Cash. Since these applications are proprietary, it is very difficult to assess their security. However, we can state that in order to offer a protection against malicious applications two main trends are followed by such major companies.

The first trend shows a growing interest in the use of Secure Elements (SE). A SE is a tamper-resistant dedicated platform (e.g., SIM card), consisting of hardware and software, capable of securely hosting applications, storing confidential data, and providing a secure application execution environment. The SE cannot be used alone because it operates in slave mode and consequently does not control the screen of the device which makes it impossible for it to offer a secure display or user input interface. Indeed, it needs a Trusted User Interface (TUI) [16] providing a trusted path from the end-user to the SE [25]. The activation of Secure Elements is typically done once the user authenticates himself (e. g., entering a PIN or a fingerprint) through a TUI. So one possible approach would be to simply implement a full existing solution on the SE. This solution has the advantage of its simplicity: it inherits the security guarantees of an existing, well understood and already widely deployed protocol. However, implementing a full protocol on a SE has a huge impact on the secure elements in terms of implementation and certification costs. It is difficult (and therefore costly) to maintain and update the solution: any update implies to revise the SE content through erase and write operations on the non-volatile memory, which supports a limited number of write and/or erase operations. In contrast, it is way simpler to update an application on a mobile platform. Moreover, in order to guarantee the security offered by the SE, any application needs to be certified before being uploaded on the SE. This is a long and costly process. This is also another reason for refraining from updating the solution; any update on the application source implies a new certification process. Finally, another important limitation is the memory size of a SE. The bigger the application is, the more space is required on the SE, which implies more expenses on the SE part, plus, SEs do not support too intensive computations.

The other main trend relies on Host Card Emulation (HCE) solutions which get rid of the device-based SE and provide a full software application. Two main options can be observed on the market nowadays. Some HCE-based applications deport the sensitive operations (i.e. what is executed by the SE) on the cloud. The main drawback of such approach is that the user needs to be connected to the cloud (through cellular connectivity or Internet) in order

to process a payment, with all the costs and availability issues it involves (e.g. roaming costs if the client is outside the country, network coverage problems...). The other kind of HCE-based applications rely on white-box cryptography designs, obfuscating application keys and cryptographic operations inside the application code itself. One first issue of this approach is the setup (or enrollment) of the user: it is difficult to pre-share personal keys between the phone and the payment service. But the main issue comes actually from the use of white-box cryptography itself. All current solutions implementing white-box-designed cryptographic primitives are broken as demonstrated by [7], which provides a generic attack called Differential Computation Analysis (DCA) which allows the extraction of key material from any published white-box implementation. The DCA Attack is significantly fast, since being automated, and requires no specific knowledge of the white-box design from the attacker.

Our objective is to propose a solution that can be used to perform payment transactions on mobile platform, even in the presence of malicious applications. We devised a new protocol that requires only a light use of SEs, that should be easy to implement and validate by certification [9]. Most of the solution is running on the mobile platform (as an HCE implementation) which offers better flexibility for developers of the payment application on the mobile platform. Our protocol has been designed with strong practical constraints in mind:

- Full compatibility with EMV-Static Data Authentication (EMV-SDA) protocol. The EMV-SDA protocol for card authentication is one of the three protocols designed by EMCCo and the default protocol implemented on merchant payment terminals. Proposing a solution compliant with existing point of sales is of high interest for a real deployment.
- Off-line payments for the customer. Even when the network is unreliable, a user should be able to pay from anywhere at any time, without having to rely on his mobile connection.
- Resistance against malicious applications. Even if a malicious application can dump the mobile platform memory (except the material stored on the SE), no fraudulent payment shall be made.
- Basic SE functionalities. We only require from the SE to store two symmetric keys and to perform very basic operations: MAC computation and counter management. We believe that such functionalities are likely to be available on most SEs and should be easy to certify.
- Ephemeral Primary Account Numbers (PANs). In contrast to standard EMV, our solution makes use of ephemeral PANs, also called *tokens* [15] (one PAN per transaction). Thanks to these tokens, our protocol provides a protection against the classic EMV-SDA replay attack thus improving the security of EMV without modifying the core protocol.

The fact that we use ephemeral PANs follows the proposition made by EMV [15] of using tokens instead of the PAN. However, this specification focuses on the requirements regarding the environment and infrastructure of tokenization for better scalability in existing payment networks. It does not provide details about the actual mechanism behind token provisioning and token-based payment. To the best of our knowledge, our protocol is the first open specification of a tokenized version of EMV, which is of independent interest.

Our protocol is composed of two phases. When the user is connected, during the first phase, he may provision tokens (typically 5 to 10 tokens) for future payments. These tokens are stored encrypted on the mobile platform. Then, during the second phase, when the user wishes to pay, he may use one of his tokens, using his SE to decrypt one of the tokens his device holds. The SE releases the key associated to the token only once the user has authenticated himself to his mobile platform and approved the payment, through the *Trusted User Interface* (TUI), which we assume to be reliable. Interestingly, when a token is decrypted, it is already linked to the merchant. Also, the transaction amount is approved by the user. The merchant's ID and the amount can no longer be modified. Moreover, even if such a decrypted token is stolen (for example if a malicious merchant cancels a payment in order to obtain a second token), then the impact of such an attack is limited. As soon as the user successfully processes an honest transaction with another (fresher) token, all previous tokens are immediately invalidated. This is realized through the use of a counter inside the SE.

To analyze our protocol and prove our security claims, we provide a formal model of our payment protocol using TAMARIN [22], a recent tool dedicated to protocol analysis and the first one that can handle counters. We formally prove several important security properties using TAMARIN. In particular, we show that even if an attacker can fully control the user phone (at the application level), he cannot trigger any payment without the user consent. In the extreme case where the attacker can decrypt a token (e.g. bypassing the TUI), then the token can only be used a limited amount of time, until the user processes to another payment. These properties are formally stated in TAMARIN and most of them can be automatically proved, except for two of them that required some user interactions due to the use of counters.

This protocol has been conceived in close collaboration with a major telecommunication company and a prototype of a mobile payment application based on this protocol has been developed within the company.

Related work. To our knowledge, our protocol is the first proposition of a payment protocol relying on the use of a SE, at least among existing open specifications. Moreover our protocol is fully formalized and proved. A security proof of the original EMV protocol has already been provided, for example in [10]. However, this analysis assumes the user device to be secure (because it is a credit card), which is no longer reasonable with mobile platform. In terms of security

proofs, the main difficulty of our protocol is the presence of counters. [8] provides a model and a proof of a key-management protocol run on a secure hardware but does not need counters. [21] provides a methodology to analyse protocols with counters in TAMARIN and applies it to the Yubikey. However, this is not a payment protocol and our protocol is more complex (more actors and more steps).

Our protocol relies on the use of ephemeral PANs as proposed by EMV [15]. Therefore, the general idea of using tokens instead of the real PAN of the user was already proposed. However, we are not aware of any (publicly available) concrete instantiation of a tokenized version of EMV. For example, whether or not a token shall be re-used is left to the designer's choice. Our protocol forms a precise instantiation of the token mechanism in the context of EMV. In particular our protocol dynamically links the PAN with the merchant and the amount of the transaction, preventing the token from being used for other purposes, still preserving offline payments form the point of view of the user, which is original w.r.t. the ephemeral PAN specification [15]. Regarding the token provisioning operation itself, some modern payment applications such as Apple Pay or Samsung Pay rely on it, however the technical specifications are not available to the public. Token provisioning methods are also used in the context of One Time Password [26].

2. Presentation of the protocol

We describe here our protocol. We first start by recalling the main step of the protocol EMV-SDA.

Glossary

- CH: Cardholder
- MA: Mobile Application
- NFC: Near Field Communication
- PIN: Personal Identification Number
- POS: Point of Sale
- SDA: Static Data Authentication
- SE: Secure Element
- TE: Trusted Enclave
- TEE: Trusted Execution Environment
- TSP: Token Service Provider

2.1. EMV-SDA protocol

EMV specifications [11], [12], [13], [14] provide details on how to process to a card-based transaction. Basically, for a transaction to be valid, the merchant needs to obtain the cardholder's Primary Account Number (PAN) and the card's expiry date among other data (determined by the cardholder's bank) in order to transfer them over the payment network to the cardholder's bank. An EMV transaction rely on three main operations:

- *The user authentication:* it is an optional process allowing the user to authenticate himself to prove he agreed on the transaction. It can either be a PIN code or a signature.
- *The transaction mode:* a transaction can either be online or offline on part of the merchant payment terminal.
- *The card authentication:* three protocols define how to authenticate the card payment information: Static Data Authentication (SDA), Dynamic Data Authentication (DDA) and Combined Data Authentication (CDA).

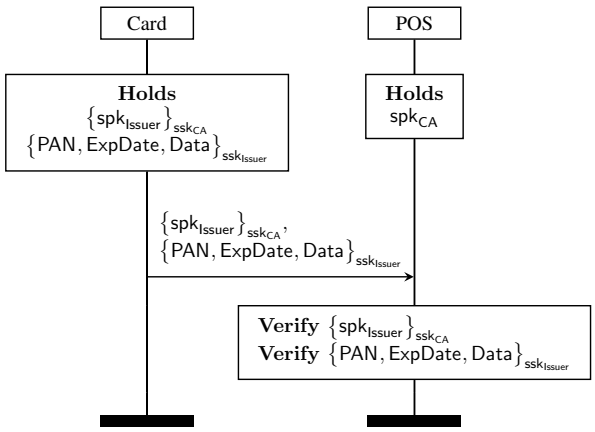

Figure 1. EMV card Static Data Authentication

The SDA protocol for card authentication is the basic one and is implemented by default on any EMV merchant terminal. It is informally described in Figure 1. During an SDA transaction, the card provides its data (PAN, expiry date and other data required by the bank) signed by the card issuer (the user's bank). The card also gives the merchant terminal the signing public key of the card issuer, certified by a Certification Authority (CA). Since the terminal owns the CA public key, it can verify the authenticity of the issuer's public key and then, use this key to verify the authenticity of the transaction data provided by the card.

In our protocol specification, we do not consider some specific roles involved in the EMV ecosystem since our protocol does not change the way they act. In particular, the communications between the merchant and the acquirer remain unchanged. The only change in this ecosystem is the involvement of the Token Service Provider whose role will be defined later on this section.

We designed our protocol according to the recommendations provided by EMVco around tokenization [15]. We chose here to use ephemeral PANS. More precisely, we define a token to be a one-time-only surrogate value replacing the PAN in a transaction, which means that during a transaction, we use an EMV-compliant payment packet

(the formerly mentioned signed packet) in which the token appears instead of the PAN. Such a packet is called a *tokenized EMV payment packet* and remains verifiable in accordance to the EMV SDA protocol. A token value can only be used once to process a payment and a user shall be able to receive as many tokens as necessary. Hence, we need to lay the grounds for a token provisioning process.

2.2. Entities

Several entities are involved during our protocol and are described as follows. Our protocol does not cover the user registration part. We also rely on another entity in the payment network whose role has been defined according to EMV specification [15], the *Token Service Provider* (TSP), an entity in charge of tokens management on the payment network. The TSP makes sure an ephemeral token is correctly translated to the user's real PAN to the issuer. We assume the user is already registered to a TSP with its $\mathsf{TR_{ID}}$ (the Token Requestor IDentifier, with which the client is registered to the TSP) and that his mobile device holds the right secret symmetric keys and that the number of tokens the device receives from the TSP at each request is already settled. Under the term *mobile device*, we consider smartphones, tablets or even most recent IoT devices (like smartwatch) which could host a Secure Element.

The **Cardholder (CH)** is the owner of both the original PAN account and the device through which payments will be performed. Since we need to identify him before processing the token provisioning request or a transaction, we assume the CH holds an idenfication value $\mathsf{ID_{val}}$ he is the only one to know or able to provide. It could either be a PIN code, a biometric fingerprint, a scheme or any other identification method supported by a mobile device.

What we call the **Trusted Enclave (TE)** is in reality the combination of two security tools that are currently embedded on most of the recent smartphones: the **Trusted Execution Environment (TEE)** [16] and the **Secure Element (SE)**. The user identification will be processed through the Trusted User Interface of the TEE, so the $\mathsf{ID_{val}}$ cannot be stolen from the main OS. The $\mathsf{ID_{val}}$ has to be known only by the Trusted Enclave. In practice, the $\mathsf{ID_{val}}$ verification could be performed by either the SE or the TEE but since in both cases a secure channel is needed between the TEE and the SE, we consider this technicality out of scope. It is only after a successful identification of the user that the TEE allows requests to the SE. The SE could either be a SIM card [2], an embedded SIM [17], [18], a Secure SD-card [4] or an embedded Secure Element. It holds two symmetric keys for identification process, one for token provisioning ($\mathsf{K_{ID}}$) and another one for payment ($\mathsf{K_{Pay}}$) as well as a counter ($\mathsf{C_{CH}}$) to prevent replay attacks. The SE can increment a counter and calculate a MAC value with variables provided from the main OS of the mobile device.

The **Mobile Application (MA)** is hosted on the mobile device rich OS (the main OS of the mobile device) and is the payment application through which the CH will require new tokens or process payments. Since the rich OS is considered as untrustworthy, the MA only manages public information that does not endanger the security of our protocol. Mainly, the MA holds the user's Token Requestor Identity ($\mathsf{TR_{ID}}$), with which the CH is registered to the TSP, and the TSP public keys - one for asymmetric encryption ($\mathsf{pk_{TSP}}$) during the Token Requesting process and the other one for signature verification ($\mathsf{spk_{TSP}}$) during the Token reception and the payment processes.

Token generation and provisioning as well as payment verification and token management and storage are processed by the **Token Service Provider (TSP)**. To achieve these tasks, the TSP holds the symmetric keys ($\mathsf{K_{ID}}$ and $\mathsf{K_{Pay}}$) associated to the Cardholder's $\mathsf{TR_{ID}}$ as well as three counters ($\mathsf{C_{TSP}}$, $\mathsf{C_{Tok}}$ and $\mathsf{C_{Pay}}$). The TSP is also the owner of private keys for signature and encryption ($\mathsf{ssk_{TSP}}$ and $\mathsf{sk_{TSP}}$) and as required by EMVCo, its signature key is certified by a Certification Authority ($\{\mathsf{spk_{TSP}}\}_{\mathsf{ssk_{CA}}}$)

A user pays through a **Merchant Point of Sale (POS)**. It could be a physical terminal as well as an Internet platform for payment. The POS - or its related merchant - is identified with a Merchant ID $\mathsf{M_{ID}}$ which will be used as a payment information by the SE to sign the transaction. As specified by the EMV protocols, the merchant holds the public key of the Certification Authority ($\mathsf{spk_{CA}}$) in order to verify the TSP public key validity when handled by the mobile application. We assume the merchant to be connected, in order for the transaction to be almost instantly verified by the TSP, which should not be a big constraint since NFC POS are also usually connected.

2.3. Protocol overview

We hereby go into more details about the two main parts of our protocol. We use the following notations:

- $aenc(m, pk)$: asymmetric encryption of m with public key pk.
- $senc(m, s)$: symmetric encryption of m with secret key s.
- $\mathrm{MAC}(K, m)$: keyed-hash message authentication code of m with the key K.
- $< x_1, ..., x_n >$: the concatenation of $x_1, ..., x_n$.

2.3.1. Token provisioning request and process. During the token provisioning process (see Figure 2), the cardholder first authenticates himself through his device using his $\mathsf{ID_{val}}$. After this first authentication, and only after that, the mobile application is granted access to the SE to request the value $\mathsf{H_{ID}}$ computed from the SE symmetric key $\mathsf{K_{ID}}$ and the current value of the counter $\mathsf{C_{CH}}$: $\mathsf{H_{ID}} = \mathrm{MAC}(\mathsf{K_{ID}}, \mathsf{C_{CH}})$. This value will attest that both the right user and the right SE were involved in the request.

The mobile application can then send the tokenization request $\mathcal{M}_{tok} = aenc(< \mathsf{TR_{ID}}, \mathsf{C_{CH}}, \mathsf{H_{ID}} >, \mathsf{pk_{TSP}})$ to the TSP, encrypting it with the TSP public key. After decryption, the TSP can first check the counter value it receives ($\mathsf{C_{CH}}$) is greater than the one it holds ($\mathsf{C_{TSP}}$), with respect to a certain tolerance limit to be defined by the service. The TSP can then check the correctness of $\mathsf{H_{ID}}$, since it also holds the

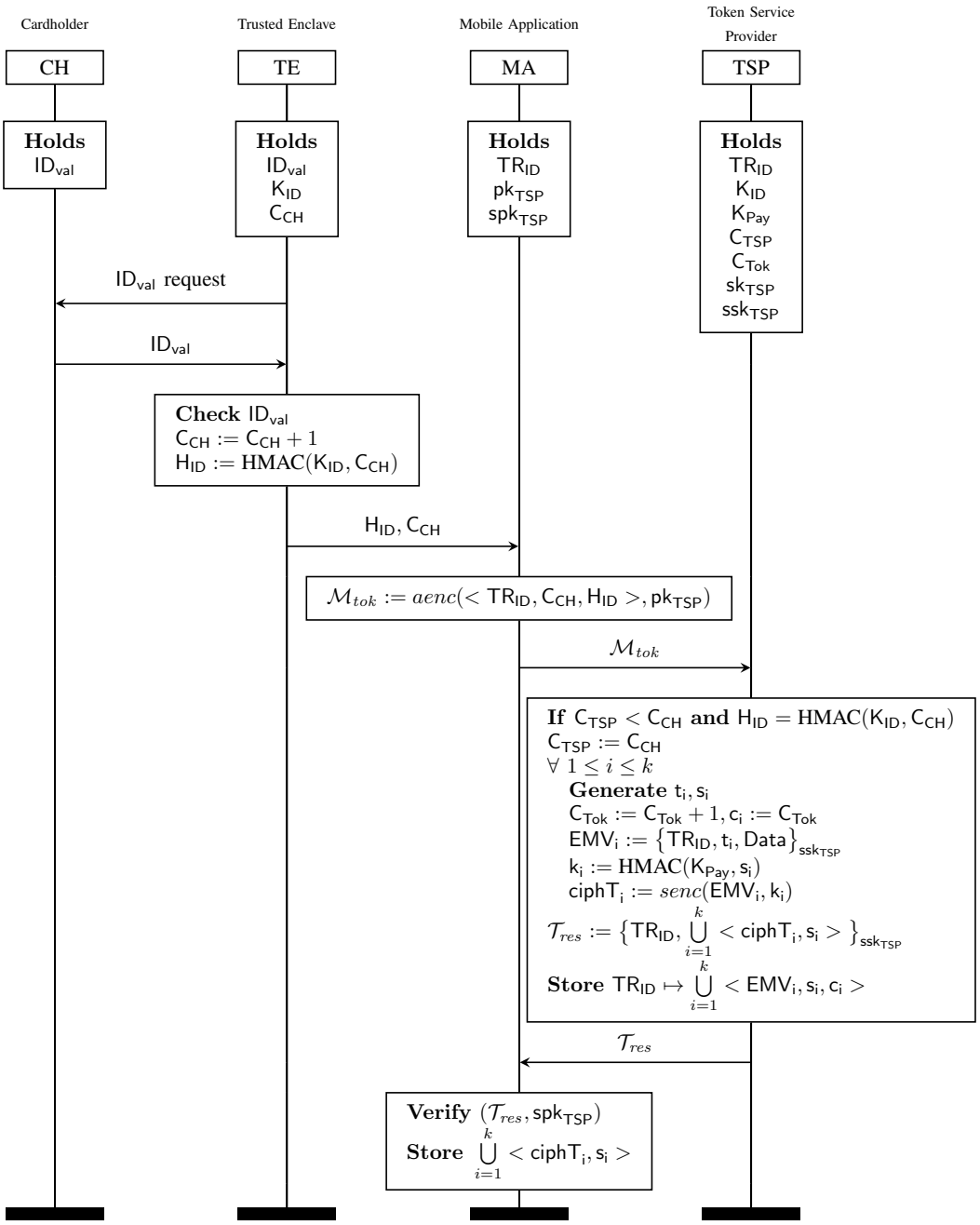

Figure 2. A Token Provisioning session

key $\mathsf{K_{ID}}$, and, if the verification is successful, increments its own counter.

After these verifications, the TSP generates k tokens $\mathsf{t_i}$ and computes k tokenized EMV payment packets: $\mathsf{EMV_i} := \{\mathsf{TR_{ID}}, \mathsf{t_i}, \mathsf{Data}\}_{\mathsf{ssk_{TSP}}}$. The Data field represents the additional data an issuer would require to validate a transaction. As required by EMVCo specifications on payment protocols, those payment packets are signed by a certified TSP signing key $\mathsf{ssk_{TSP}}$. Since each one of these EMV packets are ready-to-use payment values which will be stored on the main OS, the TSP will encrypt them with a symmetric key. For each tokenized EMV packet, the TSP generates a fresh nonce ($\mathsf{s_i}$) which will then be used with a payment key shared by the SE and the TSP to generate a symmetric

key. This key will be used to encrypt one token packet: $\mathsf{k_i} = \mathrm{MAC}(\mathsf{K_{Pay}}, \mathsf{s_i})$. Each one of the generated tokenized payment packet $\mathsf{EMV_i}$ is also associated to a counter $\mathsf{c_i}$ calculated from $\mathsf{C_{Tok}}$ on the part of the TSP. The TSP then can store the association of the $\mathsf{TR_{ID}}$ with the union of the generated $< \mathsf{EMV_i}, \mathsf{s_i}, \mathsf{c_i} >$: $\mathsf{TR_{ID}} \mapsto \bigcup_{i=1}^{k} < \mathsf{EMV_i}, \mathsf{s_i}, \mathsf{c_i} >$

The tokenization response, $\mathcal{T}_{res} = \left\{\mathsf{TR_{ID}}, \bigcup_{i=1}^{k} < \mathsf{ciph T_i}, \mathsf{s_i} > \right\}_{\mathsf{ssk_{TSP}}}$ is then sent to the mobile application, where each encrypted token ($\mathsf{ciph T_i}$) is transmitted with its corresponding nonce.

The mobile application stores those values.

2.3.2. EMV-compliant payment. Whenever a user has to pay with his device (see Figure 3), the merchant POS first has to provide it with a merchant identifier, $\mathsf{M_{ID}}$ (which could for example be an acquirer merchant identification value) and the transaction amount.

Once the user identified himself through his device, the mobile application, after choosing the oldest remaining encrypted EMV packet it holds $\mathsf{ciph T_i}$, is authorized to require two values from the SE: the encryption key $\mathsf{k_i}$ to decrypt the EMV payment packet and a transaction signature $\mathsf{T_{val}} = \mathrm{MAC}(\mathsf{K_{Pay}}, \mathsf{M_{ID}}, \mathsf{price}, \mathsf{s_i})$ which is obtained by hashing the merchant identifier, the amount of the transaction, the payment key held by the SE and the nonce corresponding to the token. This transaction signature, which will then be verified by the TSP, binds the use of a specific token to a specific merchant (so an attacker cannot pay with this token for another merchant), a specific price (which we assume to be checked by the user when he enters his $\mathsf{ID_{val}}$) and a specific SE (and thus user). With $\mathsf{k_i}$, the mobile application can retrieve $\mathsf{EMV_i}$, a ready-to-use EMV payment packet.

The mobile device can then send to the merchant POS every data required for an EMV transaction. After processing the usual EMV signatures verification the POS sends the EMV packets over the payment network to the TSP which can process the verification of the token and the transaction signature. In addition to the transaction signature verification, the TSP deactivates old tokens: $\mathsf{EMV_i}$ was associated to a counter $\mathsf{c_i}$ and the TSP holds another counter $\mathsf{C_{Pay}}$. A payment will be validated only if the actual value of $\mathsf{c_i}$ is higher than the one of $\mathsf{C_{Pay}}$. $\mathsf{C_{Pay}}$ is the counter value of the latest token validated by the TSP for a payment. If the token is too old, the payment will be refused. Otherwise the payment is authorized and the corresponding notification is sent to the POS. The TSP then updates the $\mathsf{C_{Pay}}$ value to $\mathsf{c_i}$ and deletes (or deactivates) the tokenized EMV packet from his database, as well as the older ones, preventing them to be used for a further payment. The mobile application can also erase EMV.

3. Security claims

We list the security guarantees offered by our protocol, together with our threat model.

3.1. Security assumptions and threat model

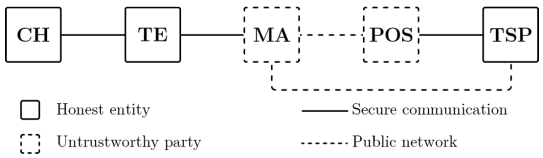

Figure 4. Communication model

3.1.1. Communication model. Figure 4 provides an overview of our communication and threat model. Firstly, we assume that the attacker has control over all public networks, in our case, the network holding the communication between the TSP and the mobile application. He can eavesdrop information and interact with all entities with values he obtained from the public network.

However, an attacker has no control over secure channels. We assume that the communication over the payment network is secure as well as every communication between a merchant POS and a TSP whether the merchant is considered as rogue or not. In the same spirit, we consider the access control to the Trusted Enclave (SE and Trusted User Interface for PIN verification) to be secure.

3.1.2. Threat model. The **TSP** is always assumed to be honest. It securely stores all the information it manages and cannot be impersonated.

A **CH** may be honest or rogue. An honest CH holds his $\mathsf{ID_{val}}$ secret and provides it to the mobile device when needed.

An **honest TE** only processes requests when the user is authenticated. We assume that whenever a user accepts to authenticate himself in order to unlock the payment for a specific merchant and value, the right price and the right merchant are displayed on the device screen through a Secure Interface. We assume a **Secure Trusted User Interface (TUI)** that is in charge of user authentication. In contrast, a **rogue TE** may share symmetric key(s) $\mathsf{K_{ID}}$ and/or $\mathsf{K_{Pay}}$with the attacker.

We consider the **MA** to be either **honest**, meaning it does not provide anything else to the public network than what is specified by the protocol or **untrustworthy** in which case, it shares its data with the attacker.

A **honest POS** behaves as prescribed by the protocol. A **rogue POS** shares its data with the attacker and behaves arbitrarily. For example, it try to input any payment value he receives from the attacker without processing the EMV verification. We allow CH to interact with both honest and rogue POS.

The **payment network** (including the merchant's acquirer and the user's issuer) is an abstract entity that we consider to be honest. Whenever a POS, rogue or not, tries to communicate with a TSP, it must be correctly authenticated through the payment network.

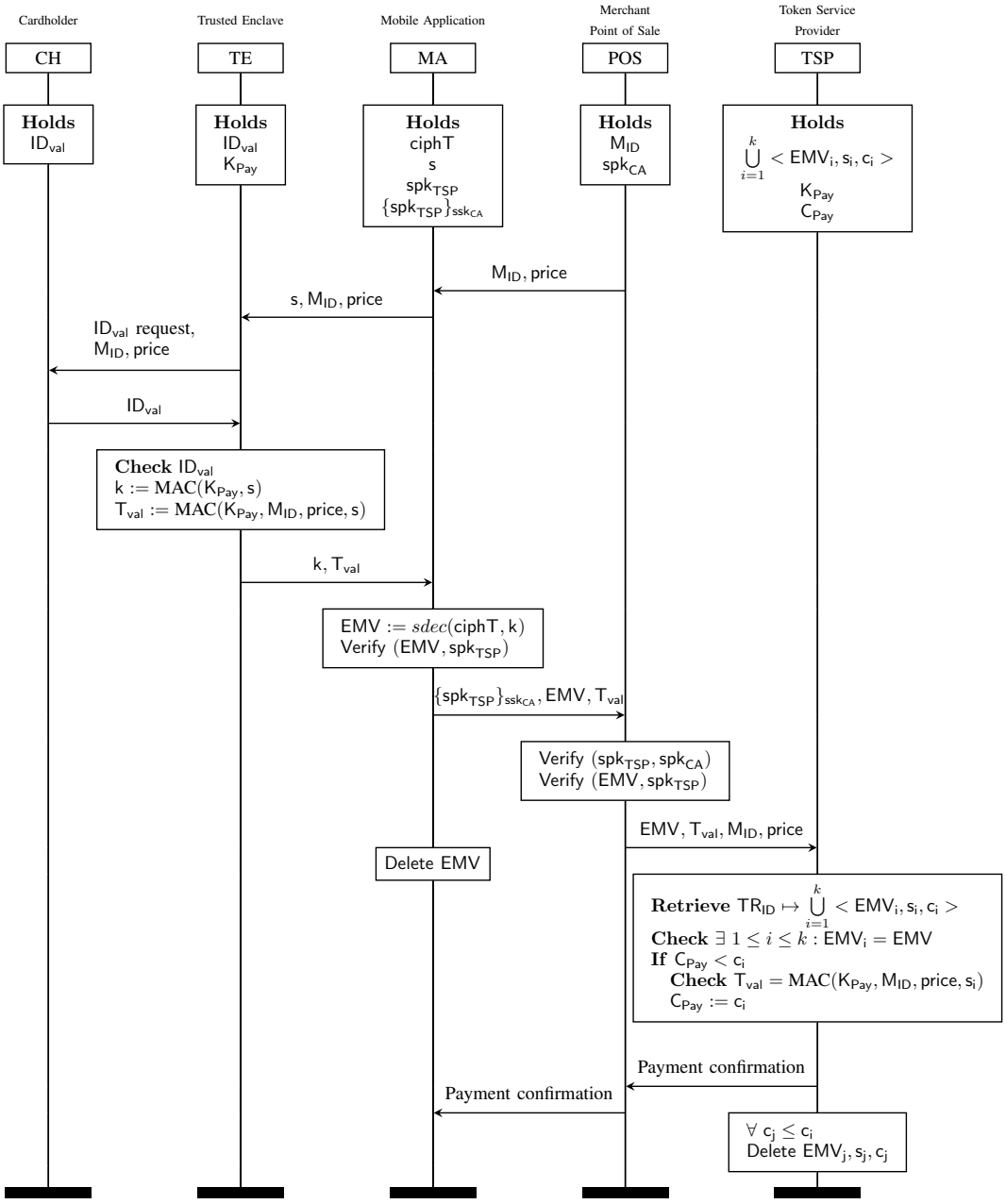

Figure 3. A Token-Based Payment

3.2. Security properties

We list here the five main properties guaranteed by our protocol. The two first ones form the natural agreement properties: whenever a transaction is made, the cardholder did consent to it and similarly, whenever a merchant accepts a transaction, he is guaranteed to be paid. The third property states similarly an agreement for the provisioning part: whenever a token is provisionned, the cardholder did initiate the request. Finally, the two last properties are further security guarantees offered by our system, in case a decrypted token is leaked.

3.2.1. Mandatory transaction agreement by the user. Each time the TSP validates a transaction between a cardholder CH and a merchant M for an amount p, then CH

must have initiated the payment request (thus agreed to it) of amount p to the merchant M by authenticating himself through the TE.

3.2.2. Merchant payment assurance. Whenever a merchant M is notified that a transaction of amount p has been validated, the TSP validated the payment of amount p to the merchant M and sent the notification.

3.2.3. Injective Token Provisioning. Each time a token is generated by the TSP for a given cardholder CH, a given key $\mathsf{K_{ID}}$, a given counter value $\mathsf{C_{CH}}$, then CH must have initiated the provisioning request by authenticating himself through his mobile (on which the SE holds $\mathsf{K_{ID}}$). Moreover, the TSP did not already generate a token for the same counter value.

3.2.4. Injective token-based payment. A token can only be used once. If a TSP validates a transaction for a specific token T owned by a client, then such a token T has not been previously used for a payment which was validated by the TSP. Interestingly, this property prevents the replay attacks against the EMV-SDA protocol.

3.2.5. Token stealing window. In case a token is stolen (for example by a merchant which maliciously claims that the transaction failed) then the consequences are mitigated by the two following properties.

- The token may be used only until a new transaction is made by CH. In other words, tokens have a limited validity.
- The stolen token may only be used for the merchant and the amount on which the user agreed upon. (Therefore a double payment can be traced.)

4. Protocol specification using the TAMARIN prover tool

There are several dedicated tools designed for analyzing and proving security protocols. Only some of them can prove security for an unbounded number of sessions: ProVerif [5], Scyther [23], as well as the recent tool TAMARIN [27]. A particularity of our payment protocol is that its security strongly relies on the use of a counter. Thus to analyze our protocol, we need an automated tool that can handle mutable states as well as an unbounded number of sessions, which excludes both ProVerif and Scyther. Therefore we chose to model our protocol using the TAMARIN prover [22], which relies on multiset rewriting rules to define a protocol. Interestingly, TAMARIN also offers an interactive prover mode, which significantly broadens the properties that can be proved.

4.1. Message theory

As usual, cryptographic messages are represented by an order-sorted algebra with the top sort `msg` and the two following incomparable subsorts `fresh` and `pub`. For each of the subsort s, we assume an infinite set of names $\mathcal{N}_s$ and an infinite set of variables $\mathcal{V}_s$. Let $\mathcal{N}$ be the union of the (disjoint) sets $\mathcal{N}_{\mathtt{fresh}}$ and $\mathcal{N}_{\mathtt{pub}}$ and $\mathcal{V}$ be the union of the (disjoint) sets $\mathcal{V}_{\mathtt{fresh}}$ and $\mathcal{V}_{\mathtt{pub}}$. Fresh names typically model cryptographic material initially unknown to the adversary such as keys or nonces while public names are known to the adversary.

Cryptographic primitives are represented as function symbols of a given signature Σ. The term algebra built over Σ, $\mathcal{N}$, and $\mathcal{V}$ is denoted $T(\Sigma, \mathcal{N}, \mathcal{V})$. The properties of cryptographic primitives are modeled through an equational theory E, that is a finite set of equations $M = N$ where $M, N \in T(\Sigma, \mathcal{N}, \mathcal{V})$. The relation $=_{\mathrm{E}}$ between terms is the smallest equivalence relation containing E and closed by application of function symbols, renaming of names and substitution of variables by terms.

Example: To model the signature scheme used during EMV payment verification, we can use the following signature and equation to define an equational theory:

$$\Sigma = \{\mathtt{sign}/2, \mathtt{pk}/1, \mathtt{verify}/3, \mathtt{true}/0\}$$

$$\mathtt{verify}(\mathtt{sign}(m, ssk), m, \mathtt{pk}(ssk)) = \mathtt{true}$$

4.2. State transition system

A protocol is represented by a labeled transition system. Each state of this system contains the adversary knowledge, the messages sent or received from the network, information about freshly generated values and the protocol-specific state. This information is represented in TAMARIN by *facts*. Formally, we assume a signature $\Sigma_{\mathtt{fact}}$ disjoint from Σ. The set of facts is:

$$\mathcal{F} = \{F(t_1, \ldots, t_n) \mid t_i \in T(\Sigma, \mathcal{N}, \mathcal{V}), F \in \Sigma_{\mathtt{fact}}\}$$

$\Sigma_{\mathtt{fact}}$ includes *persistent facts*, denoted $!F$, which are used to represent facts that are persistent, that is, are not consumed by a rule. Then a state is simply represented by a multiset of facts. A state transition system is represented by a labeled multiset rewriting rule $l - [a] \rightarrow r$ such that $l, a, r \in \mathcal{F}^*$.

Example:

```
rule 1_3_Tok_req_processing:

let < TR_ID, < C_TE, H_ID> > =
adec (M_tok, sk_TSP)
in

[In(M_tok),
 !TSP_private_keys_set($TSP, sk_TSP, ssk_TSP),
 !TSP_data(SE, TR_ID, Data, K_ID, K_Pay),
 Count_TSP(TR_ID, TE, $TSP, C_TSP),
 Fr(~s),
 Fr(~Token)]

--
[Eq(C_TE, C_TSP + '1'),
 Eq(H_ID, MAC(< C_TSP + '1', K_ID >))]
->
```

```
[Out(T_res),
 Store_Token_in_TSP($TSP, TR_ID, ~s, ~Token),
 Count_TSP(TR_ID, TE, $TSP, C_TSP + '1')]
```

The rule models the TSP behavior when answering a provisioning request. Whenever the TSP receives the request `M_tok`, it decrypts it with its private key `sk_TSP` to obtain the `TR_ID`, `C_TE` and `H_ID` values. The transition only happens if both the counter and `H_ID` are successfully checked (facts `Eq(...)`). The TSP then generates a new token `~Token` and a fresh nonce `~s`. It can then send the tokenization response `T_res` over the network (not detailed here). It will also store the token in its database (fact `Store_Token_in_TSP(...)` and increment its counter to the value `C_TSP + '1'`.

TAMARIN actually (slightly) restricts the set of labelled multiset rewriting rules that can be used in the tool but we omit this for the sake of simplicity. We refer the reader to [22] for a complete description of the syntax and the semantics.

4.3. Counter representation

Our protocol model relies on counter representation. We chose to use the built-in library *multiset* available in Tamarin to represent our counters. The counter `x` is incremented by the operation `x + '1'`. The plus sign here indicates that the element `'1'` is added to the multiset `x`. So our counters could be mathematically represented by a set of public messages: $\bigcup\{1, 1, ...1\}$. This counter representation allows comparison of values. To this end, and following the idea proposed by SAPIC [21], we use two axioms defining on one part the equality between two counter values (which is used during the Token Request processing) and on the other part an order relation between two counter values (used during the Payment validation part, to update $\mathsf{C_{Pay}}$).

4.4. Security properties specification

Lemmas are expressed as sorted first-order formulas over four kinds of atoms:

- F @ $\sharp i$, where F is a fact and $\sharp i$ is a temporal variable. It states that the fact F happens at a transition rule $\sharp i$.
- $\sharp i < \sharp j$, where $\sharp i$ and $\sharp j$ are temporal variables. It states an order between transition rules.
- $\sharp i = \sharp j$, where $\sharp i$ and $\sharp j$ are temporal variables. It states an equality of transition rules.
- $t = t'$, where t and t' are terms. It states an equality of terms.

We provide examples of TAMARIN lemmas in the next section and we refer the reader to [22] for a complete description of the syntax and the semantics of the formulas.

5. Security analysis in Tamarin

We provide a model of our payment protocol in TAMARIN, together with a formal statement of the security properties as informally described in Section 3. All model files and proof scripts are available at [1].

5.1. Protocol model

As usual, we assume the adversary fully controls public communication channels, that is, it controls the channel between the mobile platform and the merchant. In contrast, we assume that the merchant communicates with the TSP on an authenticated channel. We also assume that the TE securely communicates with the user.

The adversary initially knows all identifiers, $\mathsf{TR_{ID}}$ and the user data linked to it, the public keys ($\mathsf{spk_{TSP}}$ and $\mathsf{pk_{TSP}}$) and the counters ($\mathsf{C_{TSP}}$, $\mathsf{C_{CH}}$, $\mathsf{C_{Tok}}$ and $\mathsf{C_{Pay}}$). The adversary of course also learns all keys of dishonest TEs and mobile applications.

In order to help the TAMARIN tool and since we consider CH to be honest, we chose to merge the roles of CH and TE in our model as depicted in Figure 5. Of course, corrupted TEs have been modeled by splitting the two roles again. Similarly, we consider a one-by-one token generation per token provisioning session. This is not a limitation since a TSP can answer to arbitrarily many token requests.

5.2. Formal properties

We now state the five security properties informally defined in Section 3.2 as TAMARIN's lemmas. These properties are agreement properties. We therefore annotate protocol's rules with events (facts labeling transitions) corresponding to the different states of each agent. Where these events are placed is described in appendix, in Figure 5.

5.2.1. Mandatory transaction agreement by the user. Whenever the user is charged by the TSP, he agreed previously on the transaction.

```
lemma Transaction_agreement_by_the_user:
All TE M p #i. USER_GETS_CHARGED(TE, M, p) @ i
& not (Ex #r. CORRUPTED_K_PAY(TE) @ r)
==>
Ex CH PIN #j.
USER_AGREES_TO_TRANSACTION(CH, TE, PIN, M, p) @ j
& j < i
```

The formula expresses that whenever a user of a mobile device holding an honest TE (that is, the payment symmetric key $\mathsf{K_{Pay}}$ has not been corrupted) is charged for an amount p to pay for a merchant M at a state i (fact `USER_GETS_CHARGED(TE, M, p)`) then he has previously agreed to pay the merchant M the price p by identifying himself through the device holding the TE with his $\mathsf{ID_{val}}$ (fact `USER_AGREES_TO_TRANSACTION(CH, TE, PIN, M, p)`).

Table 1. TAMARIN PROOF RESULTS (40 CPUS, GENUINE INTEL DUAL CORE 1.2 GHZ, 200GB OF RAM)

Properties	Trust Assumptions		Execution time		Steps
	K_{ID}	K_{Pay}	wall clock time	total CPU time	
Transaction agreement by the user		NC	2min 51s	82min 50s	356
Merchant payment assurance			1min 02s	30min 09s	10
Injective token provisioning	NC		11min 23s	210min 01s	251
Injective token based payment			5min 58s	123min 40s	66
Token limited validity		NC	Manual proof 3 interactions needed		
Stolen token worthlessness		NC	7min 22s	209min 29s	645
Rogue access to mobile device		NC	Manual proof 124 interactions needed		

NC: Not Corrupted

5.2.2. Merchant payment assurance. If the merchant is notified that the transaction succeeded, he is guaranteed to be paid.

```
lemma Merchant_payment_assurance:
All TSP M p #i.
MERCHANT_IS_NOTIFIED(TSP, M, p) @ i
==>
Ex #j. TSP_VALIDATES_TRANSACTION(TSP, M, p) @ j
   & j < i
```

The formula states that whenever a merchant M is notified by the TSP that he will receive the amount p at the state i (fact `MERCHANT_IS_NOTIFIED(TSP, M, p)`) then, at a previous state j, the TSP validated the transaction (fact `TSP_VALIDATES_TRANSACTION(TSP, M, p)`) thus guaranteeing the merchant will receive the money.

5.2.3. Injective token provisioning. Each time a token is generated by the TSP, there is a (unique) corresponding request by a user. This is formalized by the following statement.

```
lemma Inj_ag_token_provisioning [use_induction]:
All TE TSP K_ID counter t1 #i.
not ( Ex #r. CORRUPTED_K_ID(TE) @ r )
& GENERATE_TOKEN_PACKET(TE, TSP, K_ID, c, t1) @ #i
==>
(Ex #j. REQUEST_TOKEN_PACKET(TE, TSP, K_ID, c) @ j
 & j < i
 & not (Ex t2 #r.
  GENERATE_TOKEN_PACKET(TE, TSP, K_ID, c, t2) @ r
        & not (#i = #r)))
```

Informally, whenever the TSP generates a token for a mobile device holding a TE (and thus, K_{ID}) and a specific counter value `c` at a state i (fact `GENERATE_TOKEN_PACKET(TE, TSP, K_ID, c, t1)`) and if the key K_{ID} is not corrupted, there exists a state j before i during which a token request was emitted from the mobile device holding the SE (and the K_{ID}) for the same counter value (fact `REQUEST_TOKEN_PACKET(TE, TSP, K_ID, c)`). Moreover, there is no other state $r \neq i$ such that the TSP generated a token for those values, which is modeled by the last part of the implication.

5.2.4. Injective token-based payment. A token owned by the cardholder can only be used once for a payment by the following statement.

```
lemma Inj_token_payment [use_induction]:
All TR_ID Tok_EMV #i.
TSP_VALIDATES_PAYMENT(TR_ID, Tok_EMV) @ i
==>
not
(Ex #j. TSP_VALIDATES_PAYMENT(TR_ID, Tok_EMV) @ j
 & j < i )
```

Intuitively, whenever the TSP validates a payment for the client identified by TR_{ID} for a specific tokenized EMV packet at a state i (fact `TSP_VALIDATES_PAYMENT(TR_ID, Tok_EMV)`), then there has not previously been such a validation from the TSP for the same tokenized EMV packet.

5.2.5. Token stealing window.

• Token limited validity. A malicious merchant could ask a client to process to payment, pretending it did not work and proceed again, in order to get one extra token that he could use later on or sell. Our protocol offers a high level of protection even in this case: not only a token can only be used by the merchant and for the amount to which the user agreed upon (Section 5.2.1) but old tokens get desactivated as soon as a payment with a more recent token is validated by the TSP.

```
lemma Token_stealing_window_token_freshness:
All c1 c2 TSP TR_ID #i.
COUNTER_VALIDITY_APPROVED(TSP, TR_ID, c2) @ i
& Sm(c1, c2) @ i
```

```
& not (Ex #j.
  COUNTER_VALIDITY_APPROVED(TSP, TR_ID, c1) @ j
  & j < i)
==>
not (Ex #r.
COUNTER_VALIDITY_APPROVED(TSP, TR_ID, c1) @ r
& i < r)
```

The fact `COUNTER_VALIDITY_APPROVED(TSP, TR_ID, c)` is activated if the TSP validates a payment with a token associated to a counter c owned by the client $\mathsf{TR_{ID}}$. So this formula states that for two tokens associated respectively to the counters $\mathsf{c_1}$ and $\mathsf{c_2}$ with $\mathsf{c_1} < \mathsf{c_2}$, whenever the TSP validates a payment with a token associated to counter $\mathsf{c_2}$) at a state i such that no token associated to $\mathsf{c_1}$ has been validated, then the TSP will never validate a payment for a token associated to $\mathsf{c_1}$.

• Stolen token worthlessness. We consider here the scenario where the attacker knows a decrypted tokenized EMV-packet and prove that, if the mobile payment symmetric key is not corrupted and if the user did not initiate the decryption of the token with his mobile device, then no payment will be made with the stolen token.

```
lemma Token_gets_stolen:
All TE EMV_packet #i.
CORRUPTED_TOKEN(TE, EMV_packet) @ i
& not (Ex #r. CORRUPTED_K_PAY(TE) @ r)
& not (Ex #l. PAYMENT_VALUES(TE, EMV_packet) @ l)
==>
not
(Ex #j. TSP_VALIDATES_TOKEN_PAYMENT(EMV_packet) @ j)
```

This property states that if the TE in the mobile device has not been corrupted (fact `CORRUPTED_TOKEN(TE)`) and has not been requested to provide payment values (fact `PAYMENT_VALUES(TE, EMV_packet)`), then, even though a token has been stolen and is fully known by an attacker (fact `CORRUPTED_TOKEN(TE, EMV_packet)`), no payment based on the `EMV_packet` token will be validated by the TSP (fact `TSP_VALIDATES_TOKEN_-BASED_PAYMENT(EMV_packet)`).

5.3. TAMARIN proof

All formal properties described in the previous section have been proven using the TAMARIN prover. Each of them was proved automatically, except two of them which required a (machine-checked) manual proof. Table 1 summarizes TAMARIN results.

It shall be noted that even if the token provisioning key $\mathsf{K_{ID}}$ is corrupted, payment security properties hold. Symmetrically, even if the payment key $\mathsf{K_{Pay}}$ is corrupted, the token provisioning property remains valid. Also, even if keys hold by the SE are corrupted, the merchant is guaranteed to be paid. Plus, a token is only valid for one payment, independently from the keys statuses, providing the user a guarantee in case his data is stolen.

5.4. Comparison of our protocol with existing EMV attacks.

The EMV-SDA protocol is vulnerable to a replay attack. Since data on the card are static (see Figure 1), if an attacker manages to get them, he can use them on a fake card in order to process payments from the user part. However, our protocol prevents such a replay attack. Thanks to our use of tokens as a one-time-only value, (still remaining compliant with EMV-SDA), our payment data is not static. In fact, the property "Injective Token-based payment" of Section 5.2.4 explicitely states that a TSP will only validate a payment with a specific token once.

Another attack on EMV [24] relies on a Man-in-the-Middle attack on Chip-and-PIN cards: a rogue card is located between the real card and the POS, the rogue card will transmit the payment data from the real card and when asked for the PIN verification, it will answer positively to any PIN introduction. However, we stated in our specification that the usage of a TEE would make it impossible to bypass the user identification, thus preventing the provisioning of payment credentials to the POS by the SE if the user identification failed. While this is a strong assumption, this requirement is part of the GSMA definition of TEE in the industry.

6. Prototype implementation

We have implemented a mobile application prototype of our protocol to provide an overview of the user experience. To compare performances, we implemented two versions of our protocol: one where the whole process runs as a software mobile application where the SE operations are emulated by the mobile application, the second version corresponds to our design, where SE operations are executed by a SIM card, which means all sensitive cryptographic operations.

The prototype was tested on a Samsung Galaxy S6 with 2.1GHz CPU and 3GB RAM. We used a SIM card provided by the Orange company. Since we do not consider the enrollment part of our protocol, all keys and counters are already provisioned to both the application and the SE (i.e. SIM card) as required by our specification. The TSP as well as the POS are both emulated by a PC. We implemented the TSP as prescribed by our protocol. For our TSP's verification signing key to be recognized by a real-life POS, we would have needed an agreement with a payment network and a certification authority. This explains why we chose to emulate the POS. The emulated POS conforms with the standards required by the payment industry [28], [11] and performs an EMV-SDA transaction process. Our code use either Java (to emulate the TSP and POS), JavaCard (SIM card) or Android (main mobile application).

We chose the PIN as a method for user identification but other authentication methods could be used. The Mobile (smartphone) and the POS communicate over NFC.

We can assess that from a user perspective, there is no real difference between a full software implementation and our solution that involves a secure element (columns 2 and 3 of Table 2). Indeed, the Secure Element can also compute a MAC in less than 1 millisecond and the round trip time between the Secure Element and the mobile application is less than 2 milliseconds. Therefore, the overhead of our

Table 2. PERFORMANCES OF OUR PROTOCOL IMPLEMENTATION (AVERAGE RESULTS OVER 50 MEASUREMENTS)

	Full software implementation	Software and secure element implementation
Token provisioning		
Token request before PIN validation	40.5ms	40.5ms
Token request after PIN validation	24.3ms	26.6ms
Server answer processing	9.05ms	9.05ms
EMV-SDA transaction		
Payment initiation before PIN validation	209ms	209ms
Tokenized EMV transaction	39.0ms	41.35ms
Individual MAC processing		
H_{ID}	0.2ms	2.5ms
k and $\mathsf{T}_{\mathsf{val}}$	0.25ms	2.6ms

protocol is negligible from a user's perspective and even our early implementation seems practical.

7. Conclusion

We proposed an open specification of an EMV-compliant protocol based on tokenization [15]. It is resistant against malicious applications that can dump the mobile device memory, since all our security properties have been proven whether the application was rogue or honest. It also requires only basic Secure Element functionalities (MAC calculation and counter management) which are available on most Secure Elements and have no impact on the Secure Element certification (e.g. Common Criteria Certification). Our early implementation demonstrates that our protocol may be used in practice.

The fact that our protocol is divided into two distinctive processes allows a user to be off-line during payment thus relieving the user to rely on his mobile connection. It also provides security properties that holds to the EMVCo standard expectations while being compliant with its most basic (and thus deployed) protocol : Static Data Authentication. However, thanks to tokenization, no replay attacks are possible, unlike the card-based protocol.

Thanks to the use of ephemeral tokens, it is no longer easy to trace a user among several transactions. As future work, we plan to show that a small variant of our protocol preserves privacy of a user w.r.t. the merchant. A more long term goal is to explore how to enhance the security of EMV-DDA and EMV-CDA using similar techniques to the ones developed in this paper.

Acknowledgments.

We are deeply grateful to Jannik Dreier for his very patient guidance through the TAMARIN tool. We also would like to thank the anonymous reviewers for their detailed comments that helped us to improve the presentation.
This work has received funding from the European Research Council (ERC) under the EU's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC) and the ANR project SEQUOIA ANR-14-CE28-0030-01.

References

[1] TAMARIN protocol model and proofs.

[2] ETSI TS 102 221. Smart Cards - UICC-Terminal interface - Physical and logical characteristics - v.13. Technical specification, ETSI, June 2009.

[3] Smart Card Alliance. *EMV and NFC: Complementary Technologies that Deliver Secure Payments and Value Added Functionality*, October 2012.

[4] SD Association. Activating New Mobile Services and Business Models with smartSD Memory cards. Whitepaper, SD Association, November 2014.

[5] Bruno Blanchet. An Automatic Security Protocol Verifier based on Resolution Theorem Proving (invited tutorial). In *20th International Conference on Automated Deduction (CADE-20)*, Tallinn, Estonia, July 2005.

[6] Mike Bond, Omar Choudary, Steven J. Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and Skim: cloning EMV cards with the pre-play attack. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, 2014.

[7] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. Cryptology ePrint Archive, Report 2015/753, 2015. http://eprint.iacr.org/2015/753.

[8] Véronique Cortier and Graham Steel. A generic security API for symmetric key management on cryptographic devices. *Information and Computation*, 238:208–232, 2014.

[9] Common Criteria. Common Criteria for Information Technology Security Evaluation, Part 1– Version 3.1 Revision 4. Technical report, Common Criteria, September 2012.

[10] 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 Berlin / Heidelberg, 2012.

[11] EMVCo. *Book 1 - Application Independent ICC to Terminal Interface Requirements*, November 2011.

[12] EMVCo. *Book 2 - Security and Key Management*, November 2011.

[13] EMVCo. *Book 3 - Application Specification*, November 2011.

[14] EMVCo. *Book 4 - Cardholder, Attendant, and Acquirer Interface Requirements*, November 2011.

[15] EMVCo. *EMV Payment Tokenisation Specification – Technical Framework*, March 2014.

[16] GlobalPlatform. Trusted User Interface API. Specification, GlobalPlatform, June 2013.

[17] GSMA. Remote Provisioning Architecture for Embedded UICC - SGP.02. Technical specification, GSMA, October 2014.

[18] GSMA. Embedded UICC Protection Profile, Version 1.1/25.08.2015 , BSI-CC-PP-0089. Common criteria protection profile, GSMA, August 2015.

[19] GSMA. Smartphones dominating global mobile connections base. Report, GSMA, February 2016.

[20] Kount. Mobile Payments and Fraud: 2016 report. Whitepaper, Kount, 2016.

[21] Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. In *Proceedings of the 35th IEEE Symposium on Security and Privacy*, SP '14. IEEE Computer Society, May 2014.

[22] Simon Meier, David Basin, Benedikt Schmidt, Jannik Dreier, Ralf Sasse, and Cas Cremers. Tamarin prover for security protocol verification - Project Page.

[23] Simon Meier, Cas Cremers, and David Basin. Strong Invariants for the Efficient Construction of Machine-Checked Protocol Security Proofs. In *23rd IEEE Computer Security Foundations Symposium*, pages 231–245, Los Alamitos, USA, July 2010. IEEE Computer Society.

[24] 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*, 2010.

[25] NIST. Security and Privacy Controls for Federal Information Systems and Organisations - Revision 4. Special publication, NIST, 2013.

[26] N. Popp. Token provisioning, June 16 2009. US Patent 7,548,620.

[27] Benedikt Schmidt, Ralf Sasse, Cas Cremers, and David Basin. Automated Verification of Group Key Agreement Protocols. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 179–194, Washington, DC, USA, 2014. IEEE Computer Society.

[28] International Standard. *Financial transaction card originated messages - Interchange message specificationsy*, 2003.

[29] Els Van Herreweghen and Uta Wille. Risks and Potentials of Using EMV for Internet Payments. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 18–18, Berkeley, CA, USA, 1999. USENIX Association.

Appendix

Figure 5 describes the protocol model we used with the TAMARIN prover as well as the events mentioned in Section 5.2

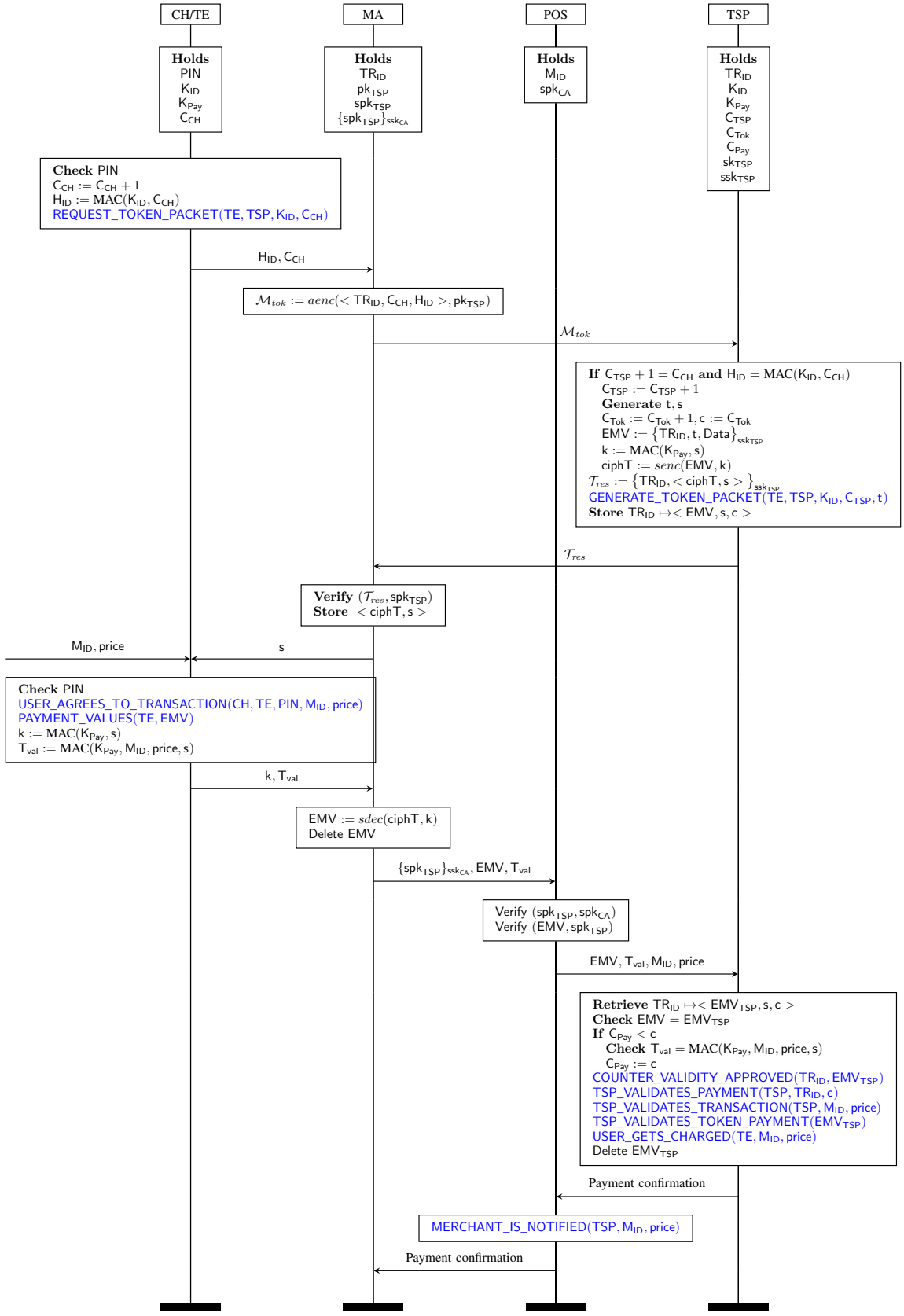

Figure 5. Our TAMARIN model